2021-06

TACTICAL APPLICATION OF MACHINE LEARNING TECHNIQUES FOR ANALYZING AUDIT RECORD GENERATION AND UTILIZATION SYSTEM (ARGUS) DATA TO DETECT BOTNET TRAFFIC

Ross, John T., II; Males, Nathaniel J.

Monterey, CA; Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

TACTICAL APPLICATION OF MACHINE LEARNING TECHNIQUES FOR ANALYZING AUDIT RECORD GENERATION AND UTILIZATION SYSTEM (ARGUS) DATA TO DETECT BOTNET TRAFFIC

by

John T. Ross II and Nathaniel J. Males

June 2021

Thesis Advisor:	Brian P. Wood
Co-Advisor:	Victor R. Garza
Second Reader:	Vinnie Monaco

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2021	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE TACTICAL APPLICATION OF MACHINE LEARNING TECHNIQUES FOR ANALYZING AUDIT RECORD GENERATION AND UTILIZATION SYSTEM (ARGUS) DATA TO DETECT BOTNET TRAFFIC	**5. FUNDING NUMBERS** NRP-20-N033-A
6. AUTHOR(S) John T. Ross II and Nathaniel J. Males	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) OPNAV N8/NCIS, District of Columbia	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Advancing botnet threats in cyberspace threaten the security of the Department of Defense (DOD) Information Network (DODIN) and have the potential to overwhelm the Defensive Cyber Forces' ability to provide timely assessments of network flow information due to the sheer volume of traffic. This is a problem because the DOD relies heavily on the capacity of the DODIN to command and control forces and achieve strategic objectives. This research assesses the performance of various machine learning algorithms on their ability to detect various types of botnet traffic using labeled ARGUS data. The research utilizes the Bot-IoT dataset that is composed of ARGUS files that summarize network traffic flows collected during several different botnet activities including Operating System fingerprinting, Service Scan, Data Exfiltration, and Keylogging data. The identification and categorization of botnet traffic within labeled data is a classification problem for which supervised learning methods are most appropriate. The algorithms explored are Random Forest, k-Nearest Neighbor, and Support Vector. The metrics to assess performance of the classifiers are sourced from rates of true positive, true negative, false positive and false negative. Those rates are used to calculate a score of accuracy, precision, and recall for each model on each type of botnet traffic. This research demonstrates that the Random Forest model is an effective tool to accurately classify and detect botnet traffic.

14. SUBJECT TERMS machine learning, Audit Record Generation and Utilization System, ARGUS, DOD Information Network, DODIN, command and control, C2, Tactics Techniques and Procedures, TTPs, decision tree, random forest, naive Bayes, k-Nearest Neighbor, Support Vector, network flow, network behavior, Python, Scikit learn	**15. NUMBER OF PAGES** 83
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

TACTICAL APPLICATION OF MACHINE LEARNING TECHNIQUES FOR ANALYZING AUDIT RECORD GENERATION AND UTILIZATION SYSTEM (ARGUS) DATA TO DETECT BOTNET TRAFFIC

John T. Ross II
Lieutenant Commander, United States Navy
BS, Excelsior College, 2009

Nathaniel J. Males
Lieutenant, United States Navy
BS, United States Naval Academy, 2015

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN NETWORK OPERATIONS AND TECHNOLOGY

from the

NAVAL POSTGRADUATE SCHOOL
June 2021

Approved by: Brian P. Wood
Advisor

Victor R. Garza
Co-Advisor

Vinnie Monaco
Second Reader

Alex Bordetsky
Chair, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Advancing botnet threats in cyberspace threaten the security of the Department of Defense (DOD) Information Network (DODIN) and have the potential to overwhelm the Defensive Cyber Forces' ability to provide timely assessments of network flow information due to the sheer volume of traffic. This is a problem because the DOD relies heavily on the capacity of the DODIN to command and control forces and achieve strategic objectives. This research assesses the performance of various machine learning algorithms on their ability to detect various types of botnet traffic using labeled ARGUS data. The research utilizes the Bot-IoT dataset that is composed of ARGUS files that summarize network traffic flows collected during several different botnet activities including Operating System fingerprinting, Service Scan, Data Exfiltration, and Keylogging data. The identification and categorization of botnet traffic within labeled data is a classification problem for which supervised learning methods are most appropriate. The algorithms explored are Random Forest, k-Nearest Neighbor, and Support Vector. The metrics to assess performance of the classifiers are sourced from rates of true positive, true negative, false positive and false negative. Those rates are used to calculate a score of accuracy, precision, and recall for each model on each type of botnet traffic. This research demonstrates that the Random Forest model is an effective tool to accurately classify and detect botnet traffic.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ACC	Accepted
ACL	Access Control List
APT	Advanced Persistent Threat
ARGUS	Audit Record Generation and Utilization System
Botnet	Bot Network
C2	Command and Control
CON	Connected
COTS	Commercial off the Shelf
CSV	Comma Separated Value
DCF	Defensive Cyber Force
DDoS	Distributed Denial of Service
DNS	Domain Name System
DOD	Department of Defense
DoDIN	Department of Defense Information Network
DoS	Denial of Service
Dr.	Doctor
ECO	Echo Request
FIN	Finished
FN	False Negative
FP	False Positive
Ft.	Fort
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronic Engineers
IFIP	International Foundation for Information Processing
INT	Initial
IoT	Internet of Things
IPS/IDS	Intrusion Protection System/Intrusion Detection System
IRC	Internet Relay Chat
JUN	June

LCDR	Lieutenant Commander
LT	Lieutenant
MAS	Mask Request
ML	Machine Learning
n.d.	not dated
NOC	Network Operations Center
NPS	Naval Postgraduate School
NRS	Neighbor Discovery Router Solicit
OCO	Offensive Cyber Operations
OS	Operating System
PCAP	Packet Capture
Ph.D	Doctor of Philosophy
REQ	Requested
RST	Reset
SSH	Secure Shell
SVC	Support Vector Classifier
TAM	Technology Assistance Model
TCP	Transmission Control Protocol
TN	True Negative
TP	True Positive
TST	Time Stamp Request
TTP	Tactics, Techniques, and Procedures
TUCS	Turku Centre for Computer Science
URP	Unreachable Port
USN	United States Navy

ACKNOWLEDGMENTS

Thank you to Kaylynn for all your loving support. Thank you for running the show, taking care of our dogs and home so I could stay focused on our thesis. I also want to thank Brian Wood, callsign "Woodie," and his bloody red pen for the most meticulous review of our work and keeping us on track. Without your guidance we surely would have gotten lost along the way. Thanks to Dr. John Monaco for the many course corrections that kept us focused on answering our research questions. Finally, thank to John Ross for being the most phenomenal thesis partner I could have ever asked for. Being invited to work on this research with you has unfolded as my greatest learning experience here at NPS.

— Nate Males

Thank you to Sarah for constantly being my rock. This last year has been filled with many occurrences of working on code long past midnight. I cannot overstate how much I appreciate your patience with me. Thanks to my boys, JT and Xander, for always expressing interest in my work. Your curiosity and creativity are inspiring, and I very much look forward to seeing how that manifests with your own academic endeavors. I want to thank several members of the staff at NPS, Glenn Cook, Brian Wood, Dr. John Monaco, and Dr. Armon Barton, for taking a chance on me. At the beginning of this journey, I had very little experience in the field of artificial intelligence and machine learning. Regardless, you trusted my enthusiasm and my will to learn, and I am so very thankful for that. Finally, thanks to Nate Males for placing so much trust in me. We have made quite a team and I cannot think of a project in my naval career that has made me more proud than the work that we have accomplished together with this thesis.

— John Ross

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. OVERVIEW

This research focuses on the analysis of supervised Machine Learning (ML) techniques applied to Audit Record Generation and Utilization System (ARGUS) datasets to determine the potential application of the techniques on DOD networks to detect Botnets. This research explored the optimization of feature selection on four Botnet traffic types: Service Scan, OS Fingerprinting, Keylogging, and Data Exfiltration. Subsequently, ML models were developed. The models leveraged algorithms obtained from the publicly available Scikit Learning suite of capabilities and were tailored to meet the objectives of this research. The ML models were applied to selections of features organized by importance and traffic types. After training and testing the models on each of the traffic types, the performance of each model was compared to one another.

B. MOTIVATION

The amount of network security related data is overwhelming Defensive Cyber Forces (DCFs). While additional defensive cyber forces were commissioned to grapple with the ever-increasing challenges associated with defending the DoDIN (Department of Defense Information Network), they employ tactics, techniques, and procedures (TTPs) that are reactive in nature. DCFs are not manned to maintain a network defender level of knowledge on the nuances of the configuration of all networks under their purview. The distributed placement of these additional defensive cyber forces renders them ill-equipped to respond to threats that are executing in real time. DCFs cannot rely on traditional signature-based threat mitigation measures to adequately protect the DoDIN. Nor is it feasible or cost effective to man DCFs to the level necessary to review millions of lines of log data to detect unknown threats as they occur. DCFs require capabilities that employ user friendly ML technology and techniques that help automate defense analysis processes and reduce the number of alerts that need to be investigated.

C. SCOPE

This research makes evaluations and recommendations on appropriate ML models, algorithms, and methods to apply towards the analysis of ARGUS network flows. The DOD does not currently employ capabilities or methods at Network Operations Centers (NOCs) that are able to identify unknown cyber threats. This thesis intends to provide recommendations of capabilities that could be employed to close this gap. For example, the capabilities used to counter known threats are effective if DCFs meticulously ensure that threat signatures are current which requires diligent monitoring of intelligence on emerging threats. However, this provides little to no protection against zero-day attacks and other unknown exploits for which there is no signature-based defense. This research will focus on supervised Machine Learning methods that can be employed with minimal training and expertise to detect malicious Botnet activity. The implementation of capabilities of this kind would enable DOD to detect Botnets faster than could be achieved by any current team of defensive cyber forces.

The ML models we developed may be applied as an investigative tool but will not serve the same function as a firewall, intrusion protection system/intrusion detection system (IPS/IDS), access control list (ACL), or another traditional network security device. The ML algorithms enable the analysis of network output from ARGUS for the aforementioned traffic types but will not include all performance and subprograms which enable core capabilities including processing requirements, output speed, power requirements, Command and Control (C2) requirements, and all associated software/ hardware requirements.

Initial plans for this research included conducting a user study to further explore use of ML in the practical domain. Unfortunately, time restrictions and the additional requirements encompassed within that endeavor required us to omit the objective from our research.

D. OBJECTIVES

The intent of this research is to answer three questions: (1) what operations and maintenance considerations apply to long term utilization of ML algorithms against Botnet

traffic; (2) which, if any, existing Machine Learning algorithms or techniques could be usefully applied to analyze network flow data: and (3) what caveats/limitations must be considered by consumers of alerts produced by ML analysis of network flow data? The scope of this research is limited to the analysis of ARGUS network flows.

E. ASSUMPTIONS

For the purposes of this research, it is assumed that NOCs have the capability to conduct packet captures and convert them into ARGUS formatted comma separated value (CSV) file. It is also assumed that there are sufficient resources available for the employment of the recommendations of this research. Sufficient resources include but are not limited to a Linux operating system with enough processing capability to run Python scripts and Scikit Learn modules.

F. APPROACH

The overall approach is to conduct tests of various ML algorithms on individual types of Botnet traffic. First, a publicly available dataset was used as a source to generate datasets of the four types of traffic: Service Scan, OS Fingerprinting, Keylogging, and Data Exfiltration. Three models were developed, and an iterative approach was applied to the training and test datasets to compare the efficiency of each model. Using the results of each model on each dataset, tables and graphs were developed to determine which models were best suited to identify Botnet traffic over a range of metrics and what factors contributed to their success.

G. BENEFITS OF RESEARCH

This research will consolidate and assess the successes of current Machine Learning capabilities to detect threats without relying upon malicious threat signature repositories. It will inform the design requirements of ML models that will aid in investigative analysis. It is an analysis of the application of available algorithms and methods to detect unidentified Botnets in a controlled setting analogous to the DoDIN.

H. ORGANIZATION

Chapter II provides background on Botnets, Botnet traffic, machine-learning algorithms, and historical relevancy of this research. Chapter III describes the model design. Chapter IV discusses the results of the experimentation. Finally, Chapter V provides research conclusions, findings and suggests possible avenues for future research to expand on the methodology, data, and lessons learned from this research.

II. LITERATURE REVIEW

A. BACKGROUND

Experts equipped with a broad range of tools and capabilities aid in the defense of networks against a myriad of threats. The internet environment is constantly in flux as new defensive mechanisms as well as new threats are introduced. One such tool that has been weaponized for nefarious behaviors is a Botnet. Emerging academic publications have consistently identified the need for capabilities that counter previously unidentified Botnets. Therefore, an analysis on the application of probabilistic detections capabilities towards the defense of the Department of Defense Information Network (DoDIN) is imperative. As defensive cyber forces are already oversaturated with information, additional defensive capabilities may serve to improve detection, analytics, collaboration, response, and reporting to anomalies and threats. The academic & scientific communities have made significant achievements in the development of probabilistic models, methods, and technologies supporting a broader effort to further identify and classify threats.

B. BOTS AND BOTNETS

Internet robots, commonly called bots, have existed since the late 1980s in the form of Internet Relay Chat (IRC) bots (Knecht, 2016). The earliest bots served to keep chat servers from shutting down chat rooms due to inactivity. Since then, bots have been evolved to serve a myriad of purposes from chat management to workload reduction. In 2000 the GTbot was discovered, masquerading as an mIRC client program and was used to launch various denial of service (DoS) attacks (Knecht, 2016). In 2007, a massive bot network (Botnet), named Storm, was discovered. Estimates indicated upwards of 50 million computers had been infected with malware correlating to Botnets (Knecht, 2016) and were being used for various cybercrimes. Botnets are networks of connected devices which have been infected with malicious software and can be remotely controlled through what is commonly referred to as a "Command and Control (C2) Server." In most cases, infected devices act as sleeping agents, otherwise known as zombies. Once infected, these

zombies can be activated to carry out a variety of exploits or malicious activities. For the remainder of our research, the terms bots, Botnets, and zombies refer to malicious bots.

Understanding the proliferation of Botnets aids in constructing a more complete picture as to why pose a substantial threat. Even in the early stages of the Botnet development, cyber criminals recognized the need for bots to install themselves onto client machines (Knecht, 2016). Several propagation methods for bot binary (a bot installation software) to become installed on a computer. While we will not delve into the details of the various methods employed to propagate Botnets, common methods include emails links, media devices, executables, drive-by downloads, and scanning for hosts with known vulnerabilities. Some of these methods leverage social engineering and common penetration techniques which allow the bot binary to be quickly downloaded (Khattak et al., 2014). Once downloaded, the bot establishes communication with the C2 server where it becomes part of the Botnet and can be used by the botmaster, an entity that directs the actions of bots under their control.

Botnets are ideal tools for many cybercriminals. The advent of the Internet of Things (IoT) has greatly increased the demand for Botnets as cyber hackers have recognized that many devices lack sufficient encryption, utilize low quality hardware or software, and can be easily infected to support Botnets (Agazzi, 2020). Botnets offer computing power and a level of anonymity to the botmaster. One can easily deduce how a large network of controlled devices can enable a plethora of attack vectors for savvy botmasters. Attacks such as "Distributed Denial of Service attacks (DDoS), Keylogging, Phishing, Spamming, Click fraud, Identity theft, and even the proliferation of other Bot malware" (Koroniotis et al., 2019) all become substantially more effective with the employment of Botnets. The botmaster's indirect relationship between the bots and the C2 server lays the stepping stones for avoiding detection and discover (Khattak et al., 2014). Botnets are particularly problematic because they are very hard to detect due to the malware that affects the devices root themselves at the X layer, which are deeper than the application or user layer by which most users interact (IBM Security, 2016). Once infected, the bots are subsequently controlled by a botmaster through a C2 infrastructure (Koroniotis et al., 2019).

C. OVERVIEW OF MACHINE LEARNING

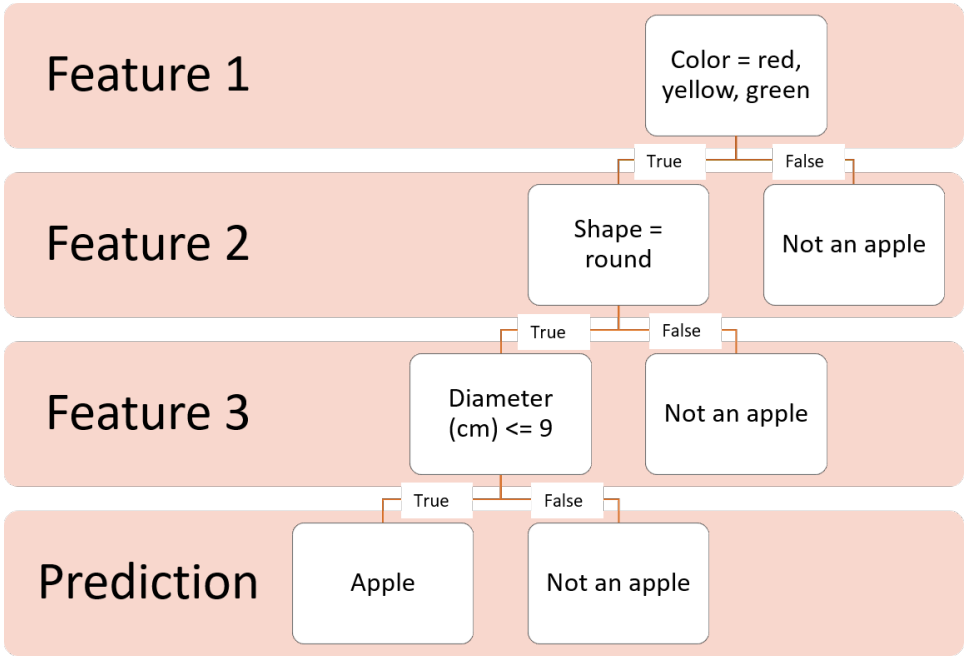
Navy Network Operations Centers (NOCs) employ tactics, techniques, and procedures that are reactive in nature: "Typical detectors are based on traditional intrusion detection techniques, focusing on identifying Botnets based on existing signatures of attacks by examining the behavior of underlying malicious activities" (Zhao et al., 2013). NOCs cannot rely on traditional signature-based threat mitigation measures to DoDIN. They require capabilities that employ user friendly Machine Learning technology and techniques that determine threats probabilistically based on *a priori* configurations.

D. ALGORITHMS

Numerous studies of Machine Learning algorithms have been successfully conducted to support the claim that Machine Learning can be an accurate tool to detect, classify, and enhance capabilities to identify Botnet attacks. "Machine Learning is an application of Artificial Intelligence wherein the system gets the ability to automatically learn and improve based on experience" (Geron, 2017). Machine Learning helps users to collect, track, and analyze data much more efficiently than if they were utilizing standard data analysis by collecting and storing in a database to be represented on spreadsheets. Classification is a subset of Supervised Learning in which data comes with additional attributes on which a prediction is made in the field of Machine Learning, classification problems require the use of a classifier. Classifiers are algorithms that can learn to detect and classify types of traffic based on their features. In our research, these classifiers will be incorporated into models written using the programming language Python. In supervised learning, examples which consist of labeled data are used to train a Machine Learning algorithm. The trained Machine Learning algorithm will then be used to make prediction on new data. When a user employs supervised learning methods, the user already has all the information required, and the Machine Learning tool can then be used to correlate features to the appropriately labeled data (Geron, 2017). In this case, the tool will predict whether the network traffic is the result of a malicious Botnet or just normal traffic. The classifiers that will be explored are Random Forest, k-Nearest Neighbor, and Support Vector.

1. Random Forest

Random Forests are composed of an ensemble of Decision Trees. A Decision Tree is a tree shaped diagram wherein each branch of the tree represents a possible decision that is made on the data. These decisions at each branch are made according to the values of the features of the data. Figure 1 offers a graphic representation of method in which a Decision Tree would use distinguishable features to differentiate between two different objects. The branch structures that perform successful classification and are also common amongst the most trees are chosen by the random forest as the final decision tree structure.



Illustrates the process by which a Decision Tree Algorithm would make a classification prediction.

Figure 1. Decision Tree Algorithm.

2. K-Nearest Neighbor

The k-Nearest Neighbor (Figure 2) is the simplest of supervised Machine Learning algorithms. It classifies a data point based on how its neighbors are classified. The k-Nearest Neighbor algorithm classifies each new instance by calculating the distance between the new instance and the distance of the labeled instances in the trained model. If

most of the labeled instances are of a certain class then new instance is classified by the k-nearest neighbor algorithm as that same class. In the example given by Figure 2 two out of three of the closest labeled instances to the new instance are green triangles, so the new instance is classified as a green triangle.

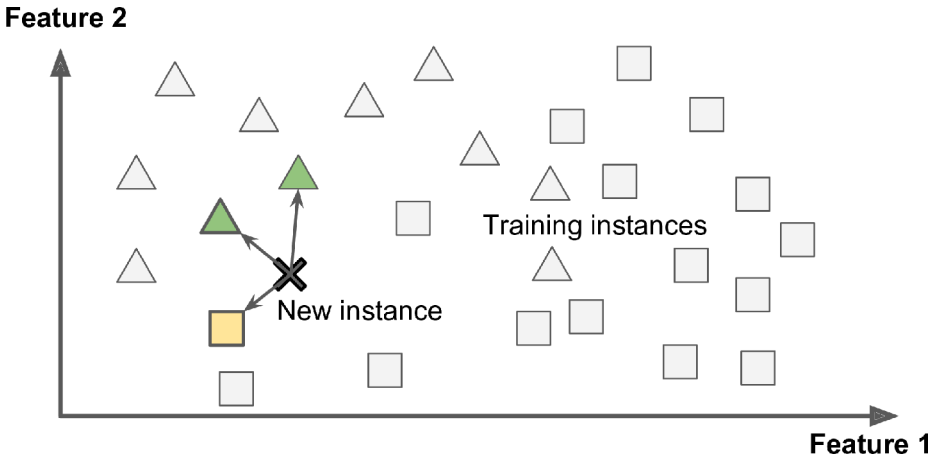


Figure 2. K-Nearest Neighbor Algorithm. Source: Geron (2017).

3. Support Vector

A Support Vector classifier groups classes of data (Figure 3) entries using their distinguishing characteristics and estimates the optimal linear separation between the groups. The Support Vector algorithm classifies each new instance by observing where the new instances reside with respect to the vector that separates the classes in the trained model. In the example given by Figure 3 the new instance is on the side of the vector populated by green triangles, so the new instance is classified as a green triangle.

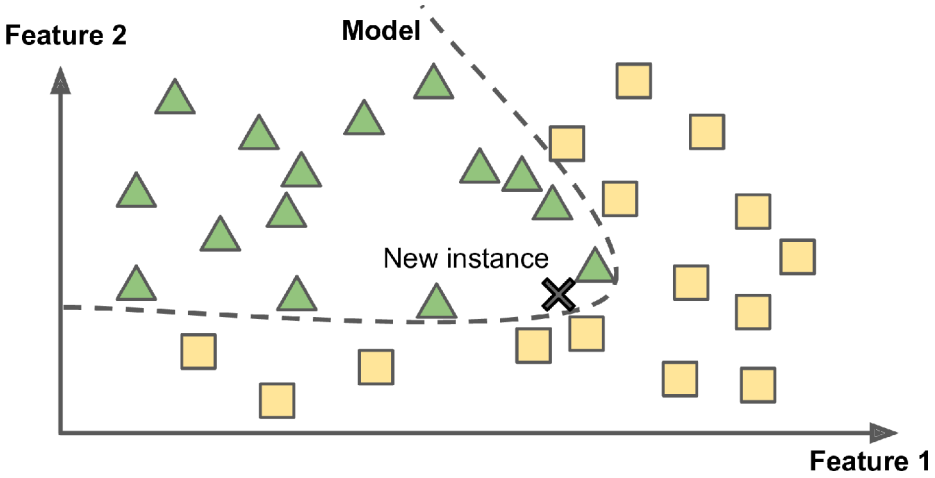


Figure 3. Support Vector Algorithm. Source: Geron (2017).

4. Approach to Model Design

There are two predominant emerging approaches to the detection of bots: detection using network flow based features and detection using graph-based features (Chowdhury et al., 2017). A network flow is a sequence of packets over some amount of time that are organized by a set of identifying features. A common software platform for converting data to network flow format is ARGUS. "ARGUS data is designed specifically for flow feature-based, as well as early and sub-flow-based, traffic analytics and classification" (openargus.org, n.d.).

The methodologies for previous research on Machine Learning vary significantly. Therefore, there is very little foundation from which to structure a uniform approach and apply those results towards NOC traffic analysis. Technology Acceptance Model (TAM) has an elegant and simple structure that fits well with the defenders of the DoDIN. Han's research, *Individual Adoption of Information Systems in Organizations* (Han, 2003), highlights the benefits of applying TAM to emerging technologies compared with other methods. TAM considers two drivers to adoption, perceived ease of use and perceived usefulness. Despite demonstrating high detection accuracy, deep inspection of the contents of network traffic is typically resource intensive and requires the parsing of large amounts of packet data which is a slow process. Therefore, the usefulness of conducting deep inspection of network traffic is progressively diminishing. Additionally, if a Botnet

employs encryption to conceal communications then it defeats the deeper inspection capabilities.

Network flow based traffic analysis is founded on the idea that there is homogeneity in Botnet traffic behavior. The features of Botnet traffic are used to distinguish them from normal traffic and further classify and and typify the traffic types (Zhao et al., 2013). This method of analysis performs classification based on features that cannot be evaded by encryption. (Zhao D. T., 2013) make the case for classification techniques with a high performance for the purposes of real time detection while simultaneously exhibiting high detection accuracy. Over the course of their individual research pursuits, they investigated several Machine Learning techniques for Botnet detection through network behavior analysis, including, Neural Networks, Support Vector classifiers, and k-Nearest Neighbor classifiers (Saad et al., 2011), and Bayes and Decision Tree (Zhao et al., 2012). There is potential to build upon the findings of their individual research efforts. *Outside the Closed World* (Sommer & Paxson, 2010) argued for the need of the system to be understood from a capabilities and limitations perspective. Upon delivery of ML capabilities to defensive cyber forces, operation of the ML capability must be as user friendly as possible to ensure that implementation is seamless and that capabilities are adopted quickly. The implementation of an ML capability should not require burdensome or overly difficult training for users to understand how to detect threats. The assessment of Machine Learning algorithms and techniques in this research will attempt to address perceived ease of use and perceived usefulness to potential users of any ML capability while also determining which existing Machine Learning algorithms or techniques can be usefully applied for detection of Botnets.

Previous researchers have used different datasets to conduct their research based upon the dataset's applicability to the goals of their research. Many have also used rates of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) for the evaluation of methods to detect Botnets. These terms are subject to various definitions based on the structure of the experiment being performed. For the purpose of this research the following are TP, TN, FP, and FN within the context of this topic. A TP is when a model correctly classifies the traffic as Botnet traffic (Salazar, 2018). A TN is when the

model correctly classifies the traffic as normal traffic (Salazar, 2018). An FP is when the model incorrectly classifies the traffic as Botnet traffic, otherwise known as a type I error (Salazar, 2018). An FN is when the model incorrectly classifies the traffic as normal traffic, otherwise known as a type II error (Salazar, 2018). Garcia (2014) leveraged the publicly available CTU-13 dataset, a dataset created by Czech Technical University in 2011 comprised of various types of analogous Botnet traffic, to assess Botnet detection methods. They used the ARGUS software suite to convert packet capture (PCAP) files to network flows. The detection methods that were used in their research consisted of a variety of approaches. Throughout their experimentation, they evaluated their methods using TP, TN, FP, and FN. Ding (2018) also assessed Botnet traffic using the CTU-13 dataset. She also used ARGUS tools as a method to output network flows which serves to segregate network traffic according to features in the network flow data. The Machine Learning algorithms used in her research were random forest and naïve Bayesian classifiers. Her research demonstrated methods that can be employed on training data to achieve a desirable balance between FP and FN.

There are three formulas that can be leveraged to encompass TP, TN, FP, and FN for the evaluation of a model's ability to classify traffic. Accuracy is the percentage of correctly classified instances and is given by $\frac{TP+TN}{TP+TN+FP+FN}$. Precision provides insight into how FPs are affecting model performance and is given by $\frac{TP}{TP+FP}$. Recall provides insight into how FNs are affecting model performance and is given by $\frac{TP}{TP+FN}$. Koroniotis et al. (2019) made Botnet predictions on network traffic using classifiers trained with the Bot-IoT dataset that they created in their testbed and subsequently made available for public use. They also used ARGUS tools to segregate network traffic according to features in the network flow data. The Machine Learning models that they used to further eliminate superfluous features in their training data consisted of Support Vector, Recurrent Neural Network and Long-Short Term Memory Recurrent Neural Network.

It is difficult to compare one set of results to another as many studies on Botnet detection apply their varying methods and metrics to different datasets which were collected in structurally diverse testbeds. In many cases, the datasets are private or only

available upon approval (Garcia, 2014). However, the aforementioned results are similar in that the classifications were made based on the characteristics of the selected features on the observed network flows. While detection methods that rely upon network flow as an input are popular, they have received criticism.

Botnet Detection using Graph-based Feature Clustering, highlights that "network flow traffic features rely on computing statistical features of flow traffic. As a result, these methods only capture the characteristics of a bot's effect on individual links, rather than on the topological structure of the neighborhood/subgraph as a whole" (Chowdhury et al., 2017). They further argue that network flow-based traffic analysis does not monitor network behaviors in a holistic manner and are more likely to be evaded when attackers employ tactics that alter the behavioral characteristics of malicious traffic. If a Machine Learning algorithm is being trained with live traffic then it is vulnerable to alterations to the traffic which may lead to misclassification of network flow data (Chen et al., 2017). Using live traffic to train Machine Learning algorithms may leave it susceptible to manipulation or enable hostile agents to alter their data to appear less malicious. In contrast, *Botnet Detection Based on Traffic Behavior Analysis and Flow Iintervals* (Zhao et al., 2013) heralds the benefits of using netflow characteristics to detect Botnets. Detection methods utilizing network traffic characteristics are not subject to the vulnerabilities of encryption algorithms (Zhao et al., 2013). They are computationally cheaper compared to earlier methods that depended on deeper inspection of the contents of network traffic. Network flows can be split into characteristic time windows to enable quick detection and isolate the threat before an attack has made any serious impact.

According to *In Machine Learning for Cybersecurity: Network-based Botnet Detection Using Time-Limited Flows* (Ding, 2018), the Internet of Things (IoT) provides more platforms to launch Botnets attacks such as "smart household devices with built-in network capabilities." In order to leverage Machine Learning to address the threat posed by Botnets, an appropriate dataset must enable proper training of an algorithm to detect them. *An Empirical Comparison of Botnet Detection Methods* (Garcia, 2014) highlights the importance for datasets to include ground-truth labels, heterogeneity, or real-world traffic. It also presents a methodology by which to compare Botnet detection methods. To

the best of our knowledge, the Bot-IoT dataset from *Towards the Ddevelopment of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics* (Koroniotis et al., 2019) is currently the only one publicly available dataset that provides full range of network packet capture which better caters to verifying outcomes (Koroniotis et al., 2019). In a comparison of 12 datasets, the Bot-IoT dataset uniquely provides ARGUS data, realistic testbed configuration, realistic traffic, labeled data, IoT traces, diverse attack scenarios, full packet capture, and new generated features (Koroniotis et al., 2019). The dataset consists of 72 million records of Botnet traffic. The testbed was configured to simulate normal bidirectional traffic utilizing DNS, email, FTP, HTTP, and SSH servers. SSH, Secure Shell, is a protocol that is used to operate network services remotely and securelyacross an unsecure network. The simulated normal traffic also consists of IoT services encountered in a smart home such as weather station applications, smart appliances, motion activated lights, remotely activated garage doors, and smart thermostats (Koroniotis et al., 2019). Instead of conducting packet captures of actual Botnets, the creators of the Bot-IoT dataset simulated attacks that used Botnet characteristic. The Botnet attacks consisted of information gathering, denial of service, and information theft. The level of diversity in the testbed traffic developed by Koroniotis et al., (2019) is similar to the complex collection of traffic that an Information Systems Security Officer would analyze at a Network Operations Center (NOC).

The Bot-IoT dataset consists of over 73 million total records. The dataset partially addresses the issue of real-world traffic by using the Ostinato tool (Koroniotis et al., 2019) to generate massive amounts of realistic normal traffic along with periodically inserting traffic that makes use of services on the servers within the testbed. Ostinato is a network traffic simulation program that enables operators to simulate real traffic characteristics within a virtual environment (ostinato.org, n.d.). The issue of ground-truth labeling is addressed by ensuring that all traffic within the testbed is appropriately labelled. The large number of rows in the dataset account for the "curse of dimensionality." In the application of Machine Learning problems, the curse of dimensionality means that if the number of features used to describe the data increases, then the ability of a model to generalize will depend upon an exponentially increasing number of samples (Bellman, 1957). Using their

"10-best" feature selection, Koroniotis et al. (2019) observed the highest precision and recall using a linear support vector classifier (SVC). Their selected features do not include details of the specific source and destination. This indicates that malicious traffic may be classified based on features independent of source and destination. Some Botnets employ techniques that conceal the source of Botnet traffic (Zhao et al., 2013). Since previous works have demonstrated that varied behavior between Botnet and normal traffic can be exposed through network flow-based Machine Learning methods, this research will explore the performance of those methods on subsets of Botnet traffic. This research will seek to optimize the combination of feature selections on those subsets of the Bot-IoT dataset for the purposes of striking a balance between accuracy, precision, and recall. Additionally, publicly available methods from the Scikit Learning suite of capabilities will be trained with those features and assessed using a simplified version of the methodology of analysis used by Garcia (2014). That assessment will be compared with the initial performance of the Machine Learning methods described in Koroniotis et al. (2019) Bot-IoT paper.

Network managers across many network types lack the tools necessary to detect many modern threats. Such gaps enable threats such as Botnets to infect a network and propagate undetected until called upon to conduct attacks. Machine Learning enables us to develop new toolsets to counter problems such as Botnet detection through rapid and systemic methods of model training to detect and classify Botnets based on their distinctive features. As the environment changes, learning algorithms can be updated to identify and capture emerging threats. Current tools in use are largely signature based and detect threats that have already been seen but Machine Learning enables us to detect threats based on the features of network traffic.

THIS PAGE INTENTIONALLY LEFT BLANK

III. METHODOLOGY

A. CHOICE OF MODELS

To begin research for the application of Machine Learning (ML) algorithms in the detection of Botnets using ARGUS data, the first step was to determine the appropriate ML approach for this type of data and successfully train the Random Forest, k-Nearest Neighbor, and Support Vector models. The Bot-IoT dataset was published with labeled data. Additionally, during discovery of ML methods, unsupervised ML was continuously documented as more suitable for tasks involving unlabeled data thus eliminating it as an approach. We determined that algorithms typically used for supervised learning problems would be most appropriate. The authors of the Bot-IoT dataset (Koroniotis et al., 2019) observed the highest precision and recall when they used a linear Support Vector Classifier. In one scenario they observed perfect precision at the expense of recall. In another scenario they observed perfect recall at the expense of precision. This research will explore whether these results can be improved by applying a support vector with a polynomial kernel in lieu of a linear kernel (also known as a linear support vector). Another motivating factor in the further exploration of Support Vector Classifiers is to determine whether there is a linear separation between the two classes of network traffic, Botnet and normal. A polynomial kernel applies polynomial functions to the data that shifts data points to allow greater flexibility in finding that linear separation. Given that relatively little ML experimentation has been conducted on this dataset, a simple algorithm such as the k-Nearest Neighbor will also be explored.

B. THE DATASET AND DATA FORMAT

The Bot-IoT dataset presented data in many formats. For ease of use and uniformity, the data was converted to the CSV format. The dataset is 74 files comprised of approximately 74 million network flow records. The dataset is composed of Distributed Denial of Service (DDoS), Denial of Service (DoS), Operating System (OS) Fingerprinting, Service Scan, Keylogging, and Data Exfiltration traffic. This dataset was reduced to traffic that is less overt and more difficult to detect to engage threats that would

likely pose the greatest risk and cause the most damage to the DoDIN. As a result, DoS and DDoS traffic were removed from the scope of this research. To reduce the datasets to the four remaining target subsets of malicious traffic, Microsoft Excel functions were utilized to extract them (Figure 4). The Bot-IoT dataset includes simulated normal traffic generated with the Ostinato tool. This traffic adds realistic network flow data to the dataset to test the models against one another without creating unnecessary real-world traffic. The three models are to be developed and will each use an iterative approach to the training and test datasets to compare the efficiency of each model. Both the training and test datasets will be discussed in greater depth later in this chapter. The models will be modularly structured such that they may be interchangeably used within the Botnet classification and detection code. The code used throughout this chapter is sourced from the author's code repository on GitHub (Ross, 2020).

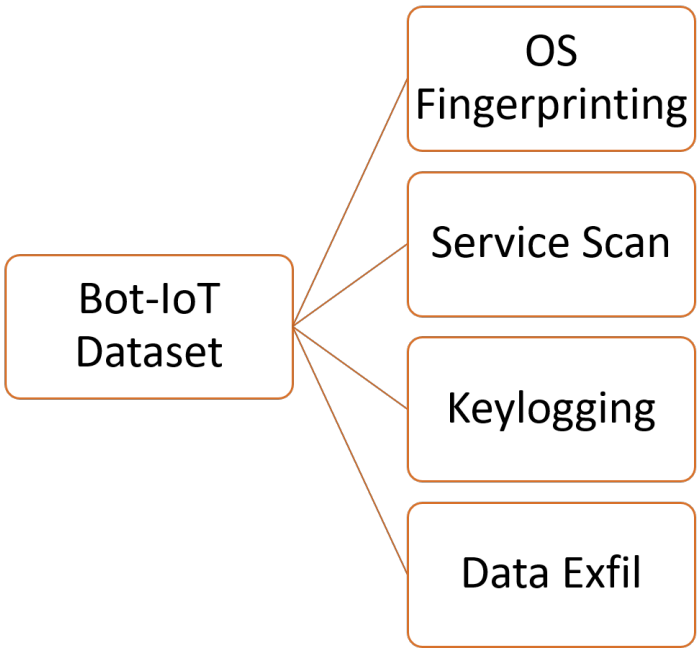


Figure 4. Four Subsets of Malicious Traffic

C. CODE STRATEGY

The purpose in designing the code in a modular format is to streamline the process of editing sections of the code without needing to create one from scratch and facilitate

future research on different classification algorithms. The format will enable data to be loaded and replaced efficiently while clearly differentiating which dataset is loaded to train the algorithm. The code structure also provides a modular interface to interchange any classification algorithm within the Scikit-learn library such that classification algorithms not explored in this research may be applied for future experimentation.

D. PROCESS FOR FINDING OPTIMAL FEATURES AND NUMBER OF FEATURES FOR EACH MODEL

To identify the number of features that optimizes model performance, a reductionist approach will be utilized. The feature reduction will be initiated beginning with all ARGUS features from the Bot-IoT dataset into the training datasets. Then, using the feature importance function from Scikit-learn's classification report package, we will be able to rank the importance of features in accordance with their importance to the model. Once importance is determined, the least important features may be eliminated one at a time until the optimal features to maximize predictive performance for each model are found.

This process continues until three features remain, and the results of the confusion matrices have been recorded. If the model had too few features, it would be overly inclusive of results akin to trying to identify a fruit such as an apple based on only color and shape; every object that meets color and shape of an apple would be included in the data and thus incur excessive false positives. After recording the results of all confusion matrices, the outcomes of each model on each dataset will be compared using scores of accuracy, precision, and recall. This entire process is shown in Figure 5.

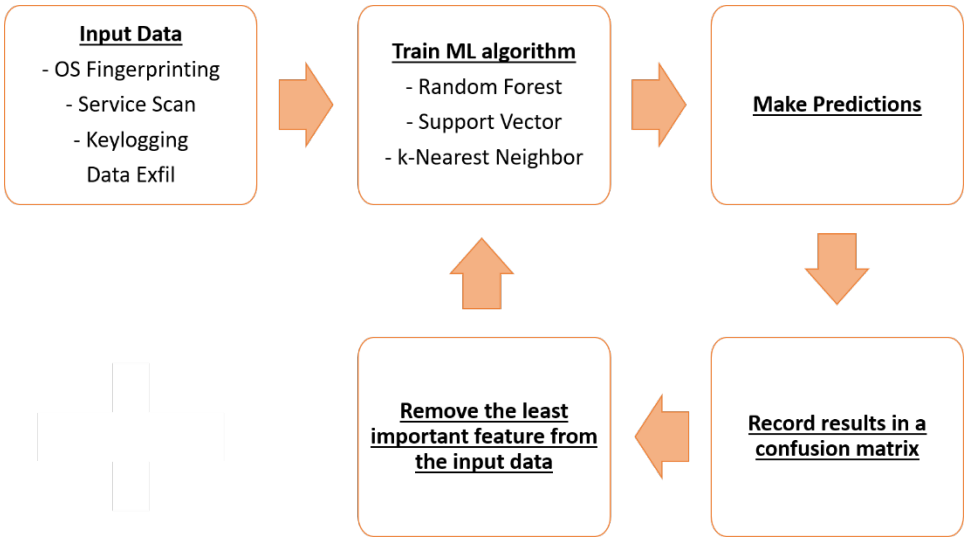


Figure 5. Feature Selection Process.

E. CODE EXPLANATION

The following in Figure 6 import commands enable the use of applicable Machine Learning algorithms and requisite mathematical functions. The commands simplify the process of importing data by optimizing memory usage for the mathematical functions, loading the desired ML algorithms, normalizing data, and formatting the dataset to be analyzed according to the applicable data type.

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

Figure 6. Importing Python and Scikit-learn Modules.

The next segment of code in Figure 7 is the applicable dataset by the local directory. The separation of datasets would enable DoDIN operators to load them individually to ultimately determine how each algorithm type performs against various categories of Botnet traffic.

```
# Loads the input data for training the ML algorithm on OS fingerprinting traffic
df = pd.read_csv('/home/john/Thesis/Bot-IoT/OS_Fingerprinting.csv')
# Or...
df = pd.read_csv('/home/john/Thesis/Bot-IoT/Service_Scan.csv')
# Or...
# Loads the input data for training the ML algorithm on OS fingerprinting traffic
df = pd.read_csv('/home/john/Thesis/Bot-IoT/Keylogging.csv')
# Or...
# Loads the input data for training the ML algorithm on Service Scan traffic
df = pd.read_csv('/home/john/Thesis/Bot-IoT/Data_Exfil.csv')
```

Figure 7. Loading Datasets.

The following segment of code in Figure 8 is particularly important in shaping the dataset to ensure that the ML algorithms can process information within the datasets. Some fields within the datasets include non-numeric values that cannot be processed by the algorithms and provide no information that would change the outcome of the models. Furthermore, some of the columns of the datasets contain no values and must be dropped to ensure that the ML algorithms can properly function.

```
# The service scan data has a persistent column resulting from a joining function when
# building the data set. This command removes that unnecessary column
del df['Unnamed: 0']

# The service scan file also has a few incorrect values that cause errors during training. This command
# cleans it from the data
df = df[df['state'] != "PAR"]

 # Drops empty columns and target column from input data
df_without_blanks = df.drop(['smac', 'dmac', 'soui', 'doui', 'sco', 'dco', 'attack'], axis=1)

# Update input data to only keep the features that will be used for training the ML algorithm
training_features = df_without_blanks.drop(['pkSeqID', 'stime', 'flgs', 'proto', 'saddr',
                                            'sport', 'daddr', 'dport', 'ltime'], axis=1)
```

Figure 8. Data Cleaning and Formatting Steps.

The next line of code in Figure 9 imports the packages required for the creation of the training and test sets to be built within the ML environment. The training sets are used to train the models to detect the various types of Botnet traffic and the test sets are used to evaluate the ability of each model to detect Botnet traffic.

```
# Import packages for creating testing and training sets from the input data
from sklearn.model_selection import train_test_split
```

Figure 9. Importing Scikit-Learn Modules.

The next line in Figure 10 defines the target variables, the dependent variables in which the algorithm will use to calculate a relationship between individual features and the target variable. Within the dataset, the target variable is the label assigned to each record. A label (the value of the row) of 0 indicates that a record is designated as normal traffic. A label of 1 indicates that the record designated is Botnet traffic. These labels are the training features that are used together to train a model to detect Botnet traffic.

```
y = df.pop('attack')
```

Figure 10. Establishing Target Variables.

The next section in Figure 11 creates the training and test set using the previously imported functions for preparing the ML environment. This command splits the dataset into a training set and a test set which both consist of their own set of features and a target variable. The segments divide the data into 80% for the training set and 20% for the test set.

```
train, test, train_labels, test_labels = train_test_split(training_features, y, test_size=0.2)
```

Figure 11. Create Training and Test Set.

Support Vector and k-Nearest Neighbor algorithms are designed such that they require numeric values to be scaled or normalized. These datasets contain a non-numeric categorical feature named state which need to be transformed into a numeric value. The most applicable method to transform the feature was OneHotEncoding whereby a column in the dataset is created for each category. The state feature has 11 categories, so 11 columns are created. The values for each record would be a one for the applicable column and zero for the remaining 10 columns. For example, if a record had a categorical value of

Connected (CON) for state then the newly created CON column would have a value of 1 and the remaining 10 columns would have a value of 10. Since the OneHotEncoding of state assigns the values of zero or one, the remaining numeric features were also normalized.

Next, the preprocessing module is loaded from Scikit-learn into the environment. Preprocessing contains the prerequisite functions necessary to perform OneHotEncoding, normalization, ColumnTransformer, and pipeline transform as shown in Figure 12. Variables are assigned as appropriate to enable transforming the original values to their desired values. Modules are also imported to enable the aforementioned functions to be placed in a sequential pipeline, and finally execute the commands that transform the categorical columns into OneHotEncoded columns (Table 1) and transform the numeric values into normalized numeric values.

```
from sklearn import preprocessing

train_num = train.drop('state', axis=1)
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Normalizer
num_pipeline = Pipeline([
    ('norm_scaler', Normalizer()),
])
from sklearn.compose import ColumnTransformer
num_attribs = list(train_num)
cat_attribs = ['state']
full_pipeline = ColumnTransformer([
    ('num', num_pipeline, num_attribs),
    ('cat', OneHotEncoder(), cat_attribs),
])
train_prepared = full_pipeline.fit_transform(train)
test_num = test.drop('state', axis=1)
num_attribs = list(test_num)
test_prepared = full_pipeline.fit_transform(test)
```

Figure 12. Data Preprocessing Steps.

Table 1. Example of One Hot Encoding.

Before One Hot Encoding

State
RST
CON
REQ
INT
URP
FIN
ACC
NRS
ECO
TST
MAS

After One Hot Encoding

RST	CON	REQ	INT	URP	FIN	ACC	NRS	ECO	TST	MAS
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	1

In order to determine which features are the most significant, it is necessary to import packages into Python capable of sorting features based on their importance ranking as shown in Figure 13. This step is important in creating the model because each combination of a dataset and algorithm within the models is hypothesized to require a different number of features for the generation of ideal results. The feature importance graphs will be products of the imported package output and will show the features based on importance for its respective dataset. The graphs will be generated for each of the traffic types to identify which features are relevant to each model and help determine a correlation between the selection of features and Botnet detection.

```
# Create variables for determining importance of features
Y_Train = train_labels
X_Train = train

Y_Test = test_labels
X_Test = test

# Import packages for calculating the feature importance ranking
from sklearn.metrics import classification_report
# Calculates feature importance ranking of the input data
trainedforest = RandomForestClassifier(n_estimators=100).fit(X_Train,Y_Train)
feat_importances = pd.Series(trainedforest.feature_importances_, index= train.columns)
plt.show(feat_importances.nlargest(26).plot(kind='barh'))
```

Figure 13. Determining Feature Importance.

The next portion of code in Figure 14 is responsible assigning the model that will be used on the loaded dataset. It is important because it determines which of the three models will be used to detect the Botnet traffic of an assigned dataset. Each of the three sections of code has settings that may be adjusted for model optimization. For Random Forest the default settings were used because they sufficiently detected and classified the Botnet traffic. For k-Nearest Neighbor, choosing the optimal value of K is not straightforward and there is not a universal standard that may be applied across datasets as the results will vary from one type of dataset to another. For the datasets used in this research a value of K=3 is used because it was determined through hyperparameter tuning to be the most efficient K value for this model. . Choosing the ideal settings for k-Nearest Neighbor requires experimentation and is dependent upon the dataset being used. For Support Vector Classifier, the default settings are used except for the kernel. Following creation of the Bot-IoT dataset, the authors conducted initial testing using a linear Support Vector classifier which produced inconsistent results (Koroniotis et al., 2019). By using a Support Vector classifier with a different kernel further insight may be granted into whether Service Scan, OS Fingerprinting, Keylogging or Data Exfiltration are linearly separable by class, Botnet and normal.

```
# Create the Random Forest model with 100 trees
model = RandomForestClassifier(n_estimators=100, criterion='gini',
                        max_depth=None, min_samples_split=2,
                        min_samples_leaf=1, min_weight_fraction_leaf=0.0,
                        max_features='auto', max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        bootstrap=True, oob_score=False, n_jobs=-1,
                        random_state=None, verbose=0, warm_start=False,
                        class_weight=None, ccp_alpha=0.0, max_samples=None)

# Or...
# Create the k Nearest Neighbor model
model = KNeighborsClassifier(n_neighbors=3, weights='uniform',
                       algorithm='auto',leaf_size=30,p=2,
                       metric_params=None,n_jobs=None)

# Or...
# Create the Support Vector Classifier model
model = SVC(C=1, kernel='poly', degree=3)
```

Figure 14. Defining Models.

This portion in Figure 15 trains the model to make predictions using the training data.

```
model.fit(train, train_labels)
```

Figure 15. Training the Model.

The next step of the code in Figure 16 makes predictions on the testing portion of the dataset based upon how the model was trained using the training data.

```
        # Make predictions on testing set

rf_predictions = model.predict(test)
```

Figure 16. Make Predictions on Test Data.

The final portion of the code in Figure 17 collects the model outputs and generates a confusion matrix which categorizes the results into four categories, true positive, false positive, true negative, and false negative.

```
        # Imports packages for making plots and graphs
from matplotlib import pyplot as plt

# Plot formatting
plt.style.use('fivethirtyeight')
plt.rcParams['font.size'] = 18

from sklearn.metrics import confusion_matrix
import itertools

# Defines function that plots the confusion matrix
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Oranges):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    Source: http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    # Plot the confusion matrix
    plt.figure(figsize=(10, 10))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, size=24)
    plt.colorbar(aspect=4)
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45, size=14)
    plt.yticks(tick_marks, classes, size=14)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.

    # Labeling the plot
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt), fontsize=20,
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.grid(None)
    plt.tight_layout()
    plt.ylabel('True label', size=18)
    plt.xlabel('Predicted label', size=18)

# Calls the function to plot the confusion matrix
cm = confusion_matrix(test_labels, rf_predictions)
plot_confusion_matrix(cm, classes=['Normal', 'Suspect'],
                      title='Traffic Classification Matrix')
```

Figure 17. Generate Confusion Matrix.

Each row in a confusion matrix as shown in Figure 18 represents the *actual traffic* present, while each column represents the *predicted traffic* (Salazar, 2018). The first row of this matrix considers normal traffic (the *negative class*) in whichTN is representative of the number of detections correctly classified as normal traffic (Salazar, 2018). FP represents the quantity of traffic incorrectly classified as suspect (Salazar, 2018). The second row within the confusion matrix considers the Botnet traffic (the *positive class*) FN (Salazar, 2018). FN represents the quantity of traffic detections incorrectly classified as normal, while TP represents detections classified correctly as suspect (Salazar, 2018). Ideally, a classifier would be characterized only be fully represented by true positives and true negatives. Visually, the matrix would be composed of nonzero values in the upper diagonal for TP and TN (Salazar, 2018).

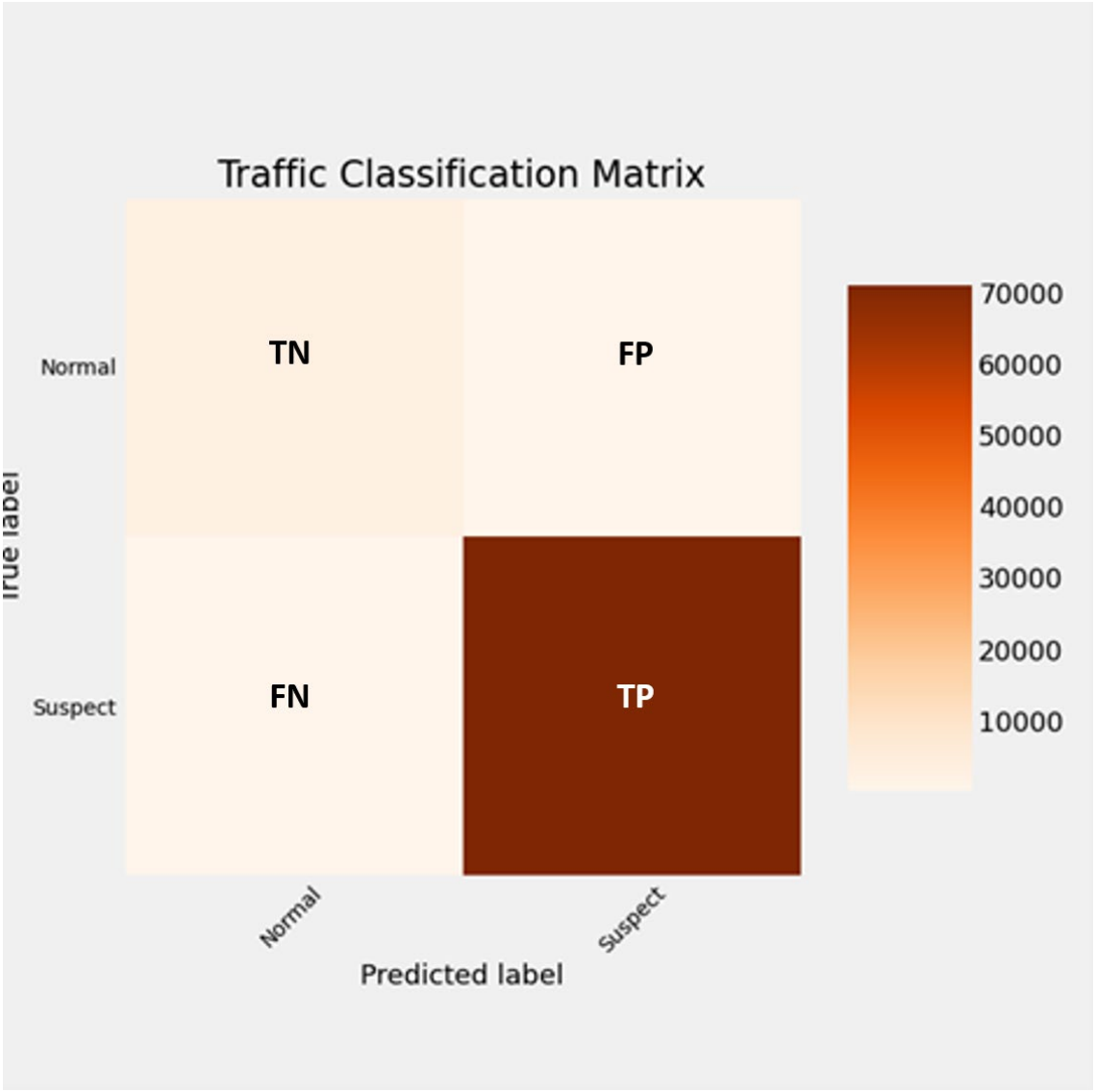


Figure 18. Confusion Matrix Example.

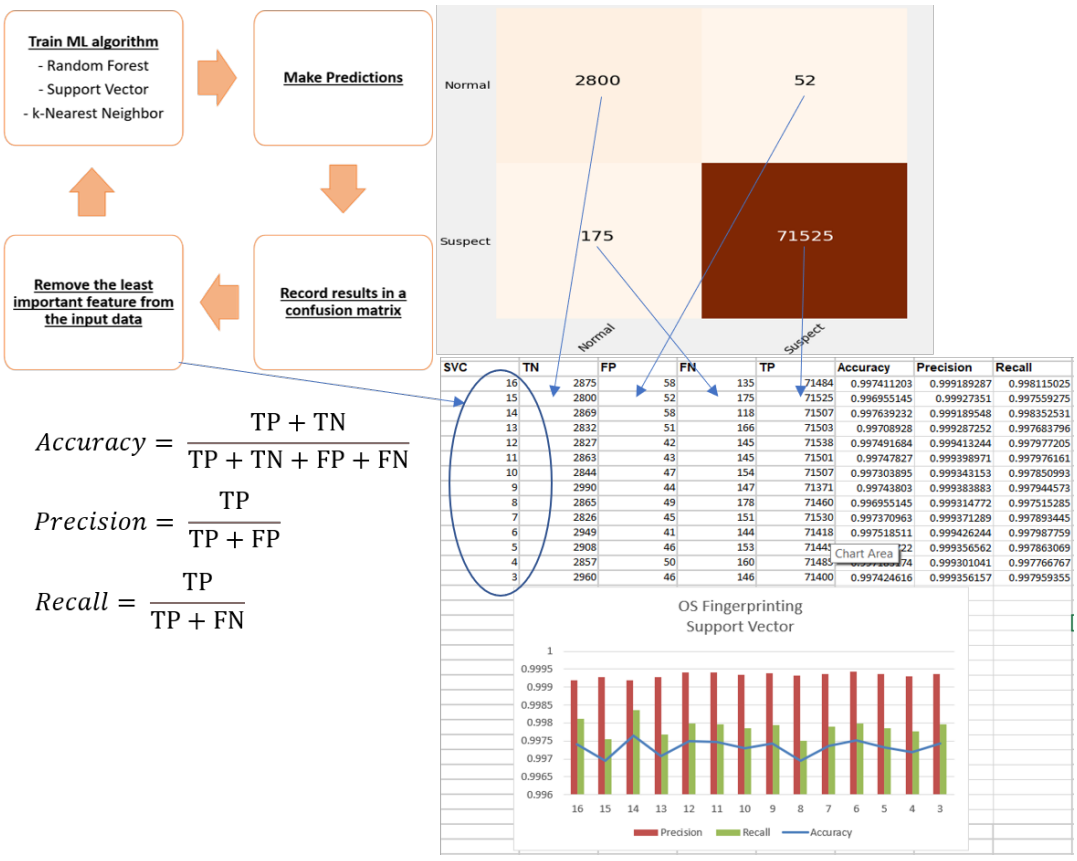
Once the confusion matrices have been generated, the results will be graphed and analyzed for patterns. If noticeable patterns are observed, then they are assessed and compared against correlation calculations, feature importance rankings, and plots of the dataset, as applicable.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ANALYSIS

A. DATA COLLECTION

Analysis of the model performances against the datasets in this study offers insight into potential applications and vulnerabilities of Machine Learning algorithms in the realm of network defense. This chapter will discuss the performance of the Random Forest, k-Nearest Neighbor, and Support Vector Classifier models against the pertinent analyzed traffic from the Bot-IoT dataset. Figure 19 below provides a graphical and elementary view of the data collection process to present a concise summary of the methodology previously discussed. It was taken from snapshots of the results of the Support Vector Classifier against the OS Fingerprinting dataset. The upper left flow chart introduces the iterative process by which the three algorithm classes were applied to the four datasets. The confusion matrix in the upper right of Figure 19 provides an example of the output from a single iteration of one of the algorithms. In the confusion matrix, the upper left contains the number of true negatives (TN), the upper right contains the number of false positives (FP), the lower left contains the number of false negatives (FN), and the lower right contains the number of true positives (TP), In this diagram. The arrows drawn from the Confusion matrix to the table illustrate the inputs of a single iteration of the algorithm as features are removed to search for optimal performance of the algorithm. Finally, the graph depicts the algorithm performance using the metrics accuracy, precision, and recall against each selection of features.



$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Figure 19. Data Collection Process

B. FEATURE RANKING

To execute the data collection process as described in Figure 19 and discussed in the methodology section of this research, feature importance for each dataset was determined using the Scikit-learn toolkit in Python. Figure 20 shows the output using that toolkit to display the varying importance of each feature as it pertains to its associated dataset. The features consist of attributes of network flow data in ARGUS format that were chosen by the authors of the Bot-IoT dataset. The three letter acronyms represent the possible transaction states of a network flow record in each dataset. Exhaustively describing the possible combinations of ARGUS attributes is outside the scope of this research. For more information reference the manual for ARGUS (openargus.org, n.d.). That varying importance provides clarity to the question of whether feature selection matters for detection of different types of Botnet traffic. When applied to an operational

setting, this research demonstrates that the feature selection plays a critical role in Botnet detection because features cannot be applied ubiquitously to a detection model; the feature selection must be tailored to the target traffic types.

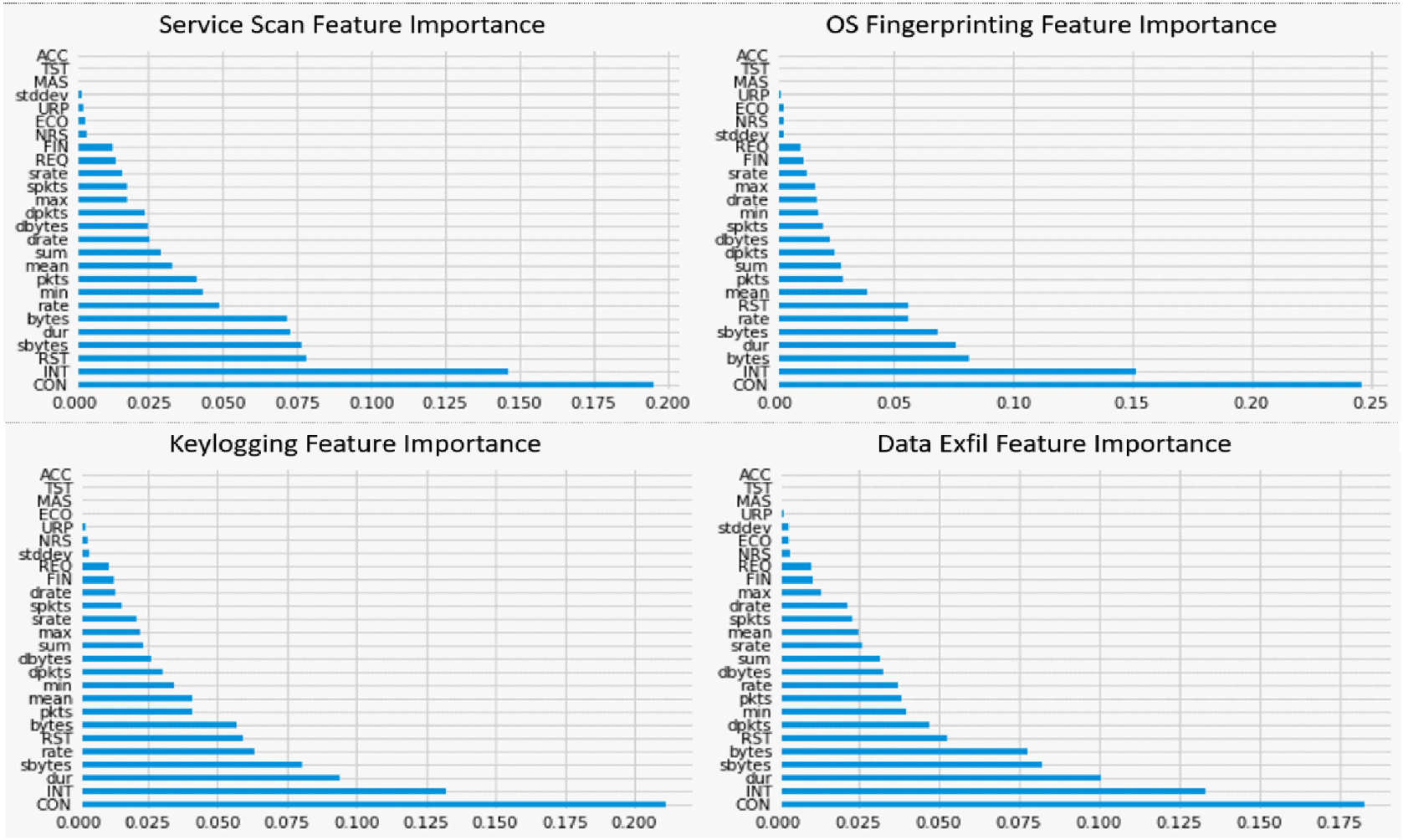


Figure 20. Scikit-Learn Feature Importance Output

C. PROCESS OF ANALYSIS

Once the dataset features had been ranked according to importance and the data collection process was complete, analysis was conducted in a row-by-row fashion, comparing how each model performed against the various types of network traffic, then a column-by-column fashion was used to compare the performance of each model against another, and concluded by conducting a holistic analysis of the models. The initial analysis was conducted with the data displayed graphically at a scale of 85% to 100% as an appropriate range to include the lower performing models. At the 85% to 100% scale, all three models performed with greater than 99% precision, accuracy, and recall for the OS Fingerprinting and Service Scan datasets. Therefore, the same analysis was conducted at a custom scale for each individual model to better capture the more subtle behavioral differences of the highest performing models. Experimentation enabled sufficient data to be captured to compare the three distinct models' ability to classify and detect malicious traffic. The results of the model's performance were assessed by comparing accuracy, precision, and recall. To reiterate from the previous chapter, accuracy is a measure of a model's percentage of correct predictions, precision is a measure of the effect of false-positives as given by the equation $\frac{\mathrm{TP}}{\mathrm{TP+FP}}$, and recall is a measure of the effect of false-negatives as given by equation $\frac{\mathrm{TP}}{\mathrm{TP+FN}}$. The overall performance of each model on each dataset is shown in Table 2. The table shows that each model did well for every metric except for k-Nearest Neighbor against the Data Exfiltration dataset, and Support Vector against the Keylogging and Data Exfiltration datasets.

Table 2. Algorithm Performance Summary.

		Random Forest			k-Nearest Neighbor			Support Vector		
		Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Service Scan	Average	0.999818	0.999867	0.999951	0.999799	0.999871	0.999927	0.998994	0.999095	0.999898
	High	0.999919	0.999952	0.999971	0.999895	0.999952	0.999957	0.999156	0.999246	0.999971
	Low	0.999552	0.999627	0.999924	0.999590	0.999656	0.999775	0.998322	0.998350	0.999861
OS Fingerprinting	Average	0.999680	0.999903	0.999764	0.999666	0.999771	0.999881	0.997328	0.999329	0.997889
	High	0.999973	1.000000	0.999986	0.999772	0.999875	0.999958	0.997639	0.999426	0.998353
	Low	0.998699	0.999553	0.999093	0.999115	0.999442	0.999637	0.996955	0.999189	0.997515
Keylogging	Average	0.998022	0.993812	0.991149	0.997730	0.992530	0.991081	0.985474	0.953463	0.939791
	High	0.999546	1.000000	1.000000	0.999092	1.000000	1.000000	0.987744	1.000000	0.996689
	Low	0.995915	0.979661	0.979094	0.995915	0.983333	0.983221	0.982751	0.900000	0.878049
Data Exfil	Average	0.999291	0.981329	0.928372	0.998851	0.946164	0.898794	0.992835	0.740068	0.338961
	High	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.995207	1.000000	0.466667
	Low	0.998631	0.913043	0.880000	0.997604	0.833333	0.789474	0.989387	0.461538	0.217391

D. GENERAL OBSERVATIONS

The four datasets used in this research were derived from the Bot-IoT dataset and separated by traffic type. The Service Scan dataset contains the most Botnet traffic with a total of 1,048,575 records, 1,046,582 being Botnet traffic. The OS Fingerprinting dataset contains the second most Botnet traffic with a total of 372,759 records, 358,275 of which is Botnet traffic. The Keylogging dataset contains the second least Botnet traffic with a total of 11,012 records, 1,469 of which is Botnet traffic. These numbers are derived from Table 3. The Data Exfiltration dataset contains the least Botnet traffic with a total of 14,602 records, 118 of which is Botnet traffic. This is reflected in Table 4 later in this chapter where further analysis on this observation is also discussed. For Service Scan, OS Fingerprinting, and Keylogging, over 90% of the traffic labeled as Botnet traffic had a Transmission Control Protocol (TCP) state of reset (RST) for its value of the state feature.

> TCP is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which support multi-network applications. The TCP provides for reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks. TCP is intended to provide a reliable process-to-process communication service in a multinetwork environment. The TCP is intended to be a host-to-host protocol in common use in multiple networks. (DARPA, 1981)

The purpose of the RST state is to reset the connection between two hosts. This means that over 90% of the model's ability to make predictions originated from the value of a single feature, TCP State RST. Based upon the observations made on the Bot-IoT dataset over the course of this research, the Kali Linux tools that were used to generate the Botnet traffic consisted of traffic with significantly more RSTs compared to the normal traffic generated by the Ostinato tool. While the Botnet traffic consists of legitimate examples of malicious traffic, the normal traffic does not seem to be representative of what one would expect in the real-world. This means that the "normal traffic" from the Bot-IoT dataset is not representative of normally observed network traffic.

> A study by Arlitt and Williamson (2005) reveals TCP RSTs are surprisingly common on the internet. Their research examined various states of TCP packets from the University of Calgary's border router over the course of one year. Roughly 15% of all TCP flows were terminated by an RST packet after the payload had already been sent in at least one direction. The reset rate was even higher for HTTP traffic, with 22% of the flows terminated by a client-side RST, and 3% by a server-side RST. In contrast, TCP RSTs occurred in less than 0.2% of records labeled as normal traffic in the Bot-IoT dataset. (Weaver et al., 2009)

The state feature for the Data Exfiltration dataset was more balanced as it was not characterized by a bias in its values. This explains the statistically significant performance disparity between each model's ability to classify Botnet traffic in the Data Exfiltration dataset as compared to the other datasets. In short, the datasets bias towards TSP state of RST enabled each model to classify Botnet traffic more effectively regarding the OS fingerprinting, Service Scan, and Keylogging subsets.

Table 3. Notional TSP State of RST Exclusion Table.

	TN	FP	FN	TP	Accuracy	Precision	Recall
OS Fingerprinting	14463	21	24174	334101	0.935092	0.999937	0.932527
Service Scan	1986	7	95171	951411	0.909231	0.999993	0.909065
Keylogging	9528	15	142	1327	0.985743	0.988823	0.903336

E. RANDOM FOREST OVERVIEW

Figure 21 shows that the Random Forest model performed exceptionally well against the Service Scan and OS Fingerprinting datasets but also performed well against the Keylogging dataset despite having significantly less malicious traffic to train the model. The Data Exfiltration dataset contained the lowest number of Botnet records on which the algorithm could be trained and did not exhibit a bias in the state feature like the other datasets, subsequently the Random Forest models performed less effectively against the Data Exfiltration dataset than the other models. Against the Data Exfiltration dataset, the Random Forest models yielded no observable patterns for accuracy, precision, or recall. In a Random Forest algorithm, each feature is used to separate Botnet traffic from normal traffic. The process is repeated until there are no features left and the dataset has been divided as much as the features will allow.

Figure 21. Random Forest Performance Summary.

F. RANDOM FOREST ON SERVICE SCAN

Figure 22 shows that Random Forest consistently achieved greater than 99.95% accuracy, precision, and recall against the Service Scan dataset. As the number of features were reduced, the performance as measured by accuracy, precision, and recall remained relatively constant and effective until less than seven features were included in the model.

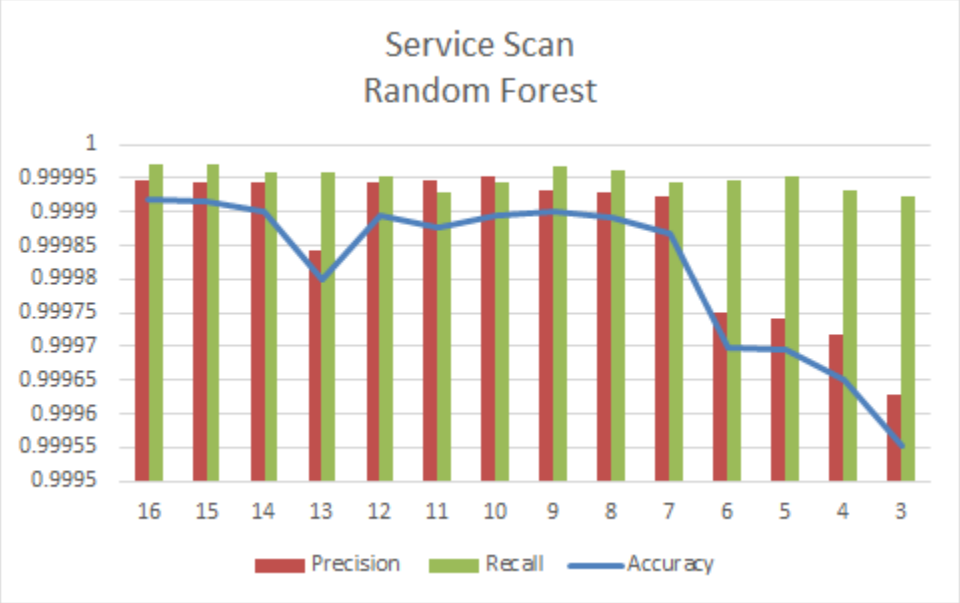


Figure 22. Random Forest Service Scan Performance Summary.

G. RANDOM FOREST ON OS FINGERPRINTING

Figure 23 shows that Random Forest consistently achieved greater than 99.86% accuracy, precision, and recall against the OS Fingerprinting dataset. When the number of features was reduced to less than seven the model performance dropped slightly, which was a behavior also observed on the Service Scan dataset.

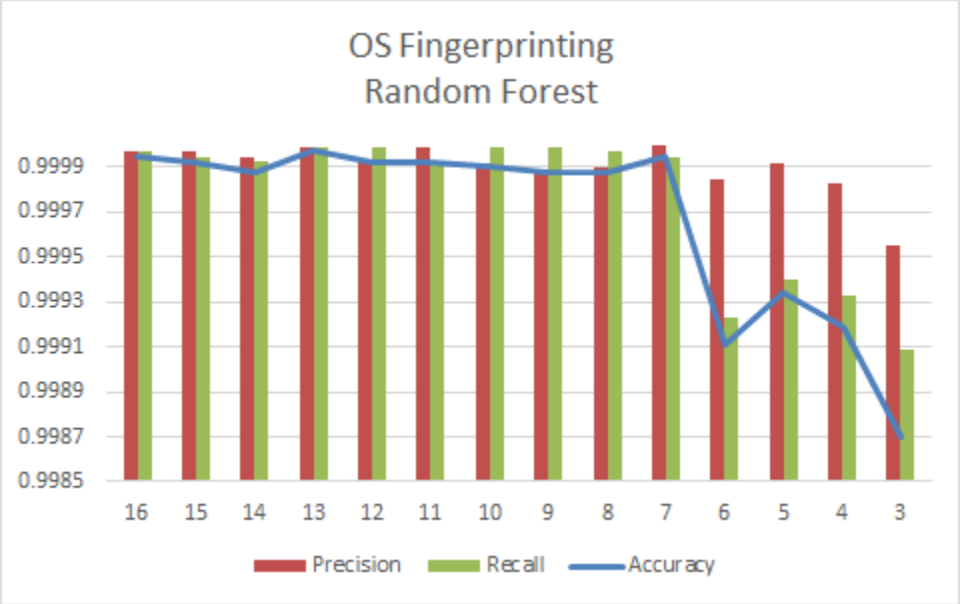


Figure 23. Random Forest OS Fingerprinting Performance Summary.

H. RANDOM FOREST ON KEYLOGGING

Figure 24 shows that Random Forest consistently achieved greater than 97.9% accuracy, precision, and recall against the Keylogging dataset. There are no obvious patterns in performance as the scores for precision and recall randomly oscillate between 97.9% and near 100%.

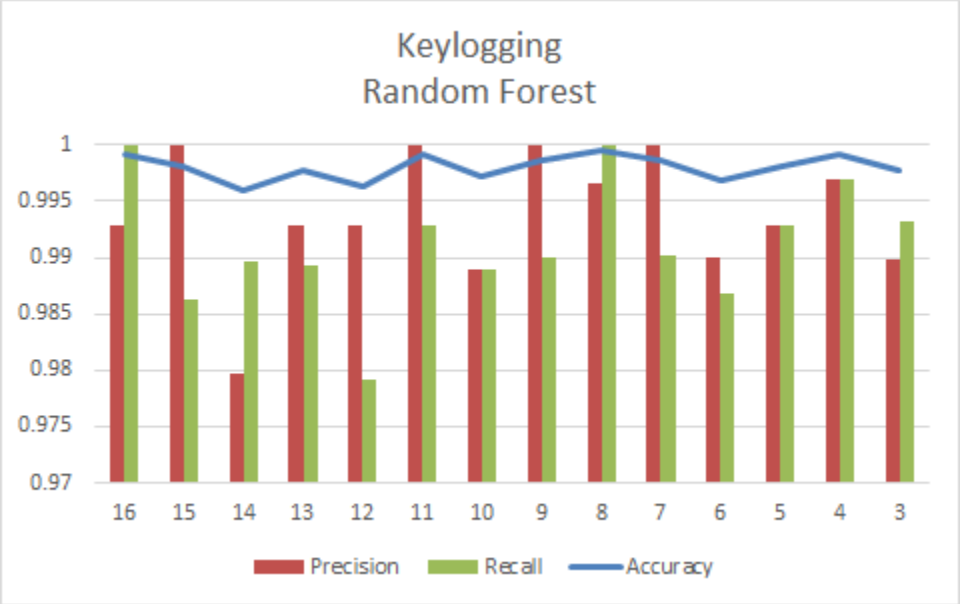


Figure 24. Random Forest Keylogging Performance Summary

I. RANDOM FOREST ON DATA EXFILTRATION

Figure 25 shows that the Accuracy metric for Random Forest against the Data Exfiltration data consistently achieved greater than 99.8%, while precision and recall achieved greater than 91.3% and 88%, respectively. There are no obvious patterns in performance as the scores for precision and recall randomly oscillate between 88% and 100%.

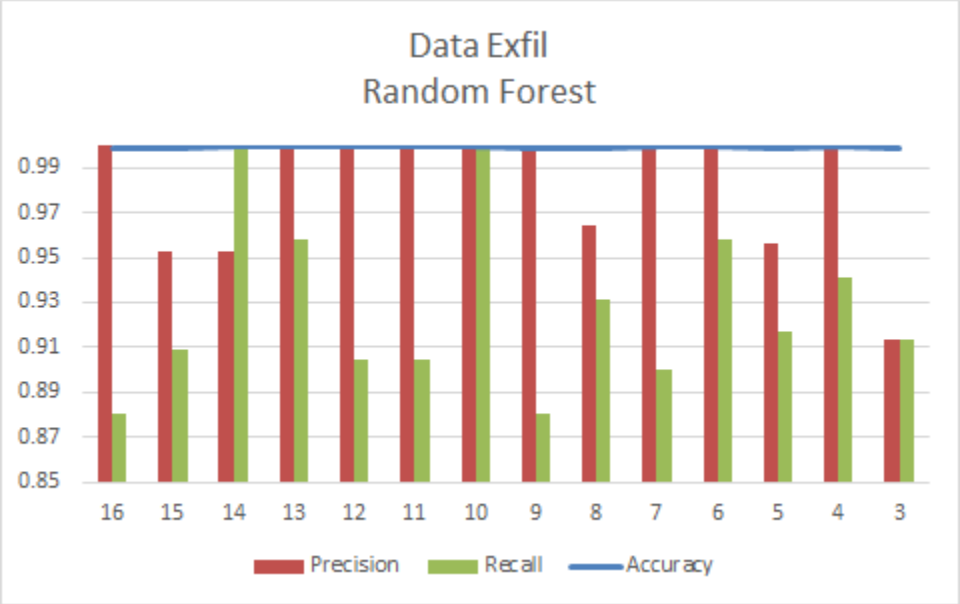


Figure 25. Random Forest Data Exfiltration Performance Summary

J. K-NEAREST NEIGHBOR OVERVIEW

Figure 26 shows that the k-Nearest Neighbor model's performance was like the Random Forest model in many respects; the model performed quite well against the Service Scan, OS Fingerprinting, and Keylogging traffic but was less impressive against the Data Exfiltration traffic. As previously discussed, the Data Exfiltration dataset contained the lowest number of Botnet records on which the algorithm could be trained and the Data Exfiltration dataset also does not exhibit a bias in the state feature as does the other datasets. While the performance on the Data Exfiltration dataset yielded the model's least impressive results, it exhibited sinusoidal patterns with points of perfect performance. In a k-Nearest Neighbor algorithm, the features are leveraged to plot groups of Botnet traffic versus normal traffic and classify new instances based upon the classification of the surrounding group. One selection of features may result in Botnet and normal traffic being grouped such that there is a higher number of false positives while another choice of feature selection may result in a high number of false negatives.

Figure 26. K-Nearest Neighbor Performance Summary

K. K-NEAREST NEIGHBOR ON SERVICE SCAN

Figure 27 shows that the k-Nearest Neighbor algorithm consistently achieved greater than 99.995% accuracy, precision, and recall against the Service Scan dataset. As the number of features were reduced, the performance as measured by accuracy and precision steadily decreased while recall remained relatively unchanged.

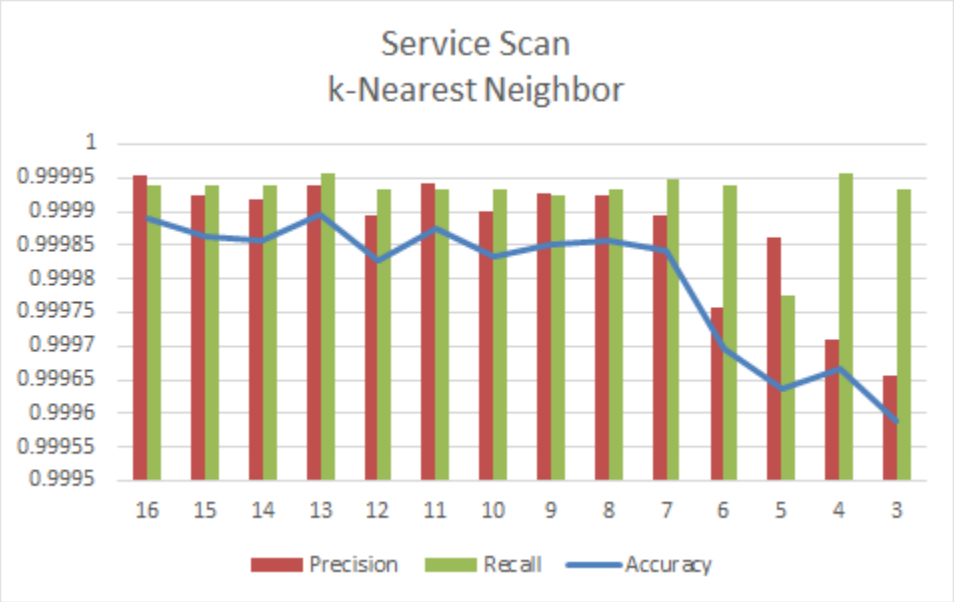


Figure 27. K-Nearest Neighbor Service Scan Performance Summary

L. K-NEAREST NEIGHBOR ON OS FINGERPRINTING

Figure 28 shows that the k-Nearest Neighbor algorithm consistently achieved greater than 99.9% accuracy, precision, and recall against the OS Fingerprinting dataset. Generally, the overall performance remained relatively consistent through feature reduction until less than five features were used. The k-Nearest Neighbor best performed on the OS Fingerprinting dataset with 7 features.

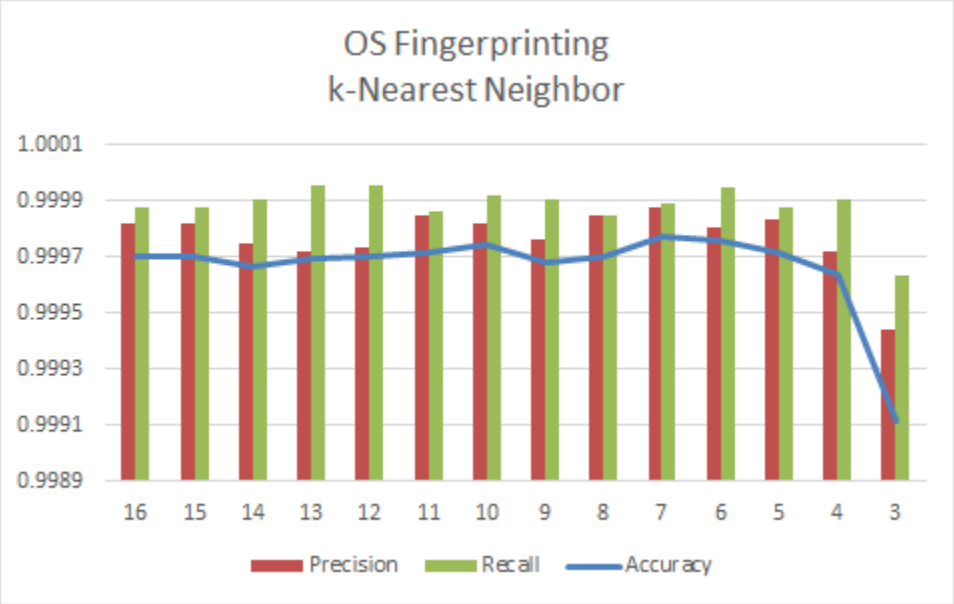


Figure 28. K-Nearest Neighbor OS Fingerprinting Performance Summary

M. K-NEAREST NEIGHBOR ON KEYLOGGING

Figure 29 shows that the k-Nearest Neighbor algorithm consistently achieved greater than 98.3% in accuracy, precision, and recall against the Keylogging dataset. There are no obvious patterns in performance as the scores for precision and recall randomly oscillate between 98.3% and 100%.

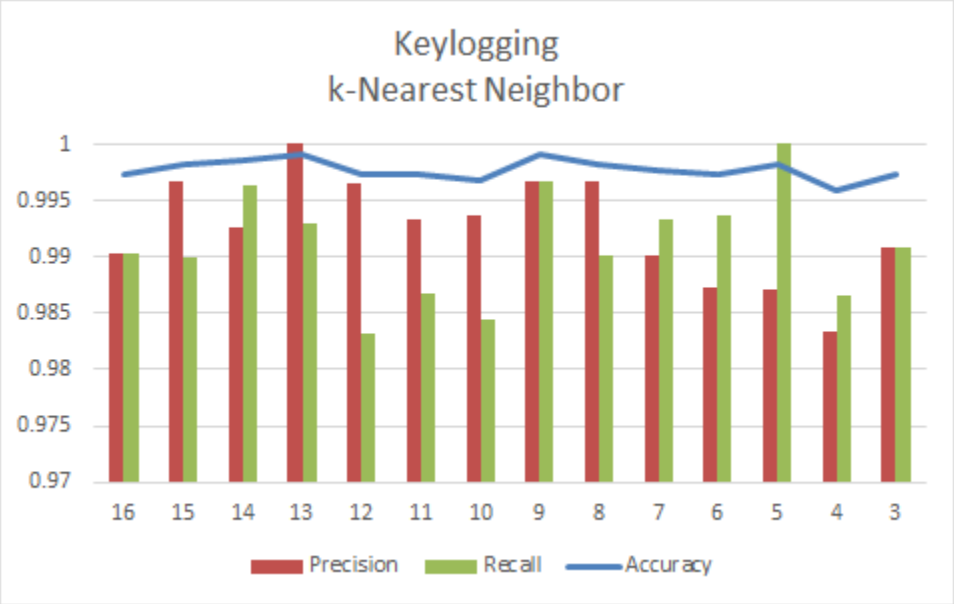


Figure 29. K-Nearest Neighbor Keylogging Performance Summary

N. K-NEAREST NEIGHBOR ON DATA EXFILTRATION

Figure 30 shows that the Accuracy metric for the k-Nearest Neighbor algorithm consistently achieved greater than 99.7%, while precision and recall achieved greater than 83.0% and 78.9%, respectively, against the Data Exfiltration dataset. There are two points at which the algorithm reaches perfect performance. Those points are when the algorithm uses 7 features and 15 features.

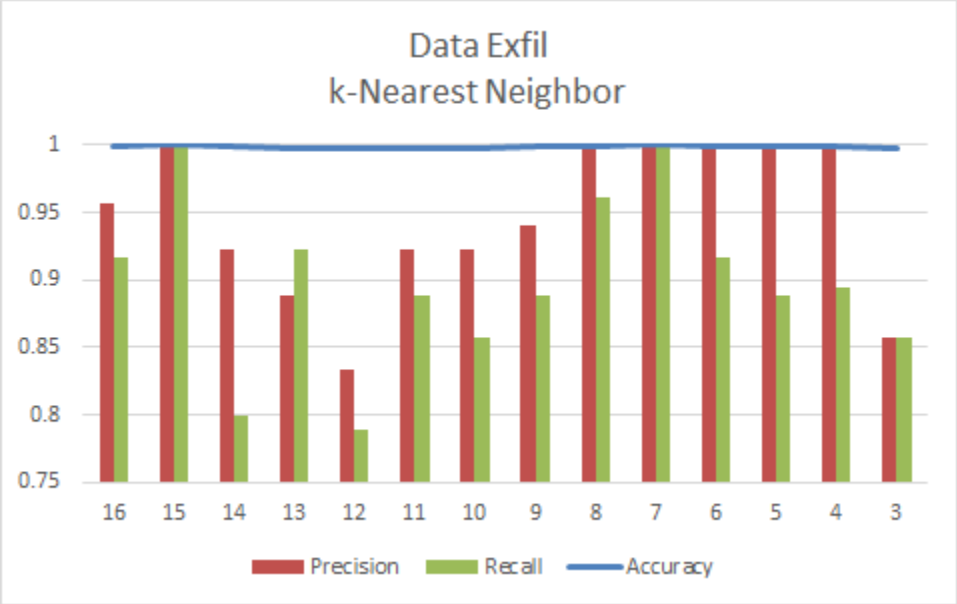


Figure 30. K-Nearest Neighbor Data Exfiltration Performance Summary

O. SUPPORT VECTOR OVERVIEW

Figure 31 shows that the Support Vector results were inconsistent and varied in accuracy, precision, and recall with each dataset. Support Vector performed well on Service Scan and OS Fingerprinting but performed erratically on Keylogging. Support Vector yielded no useful results against Data Exfiltration dataset due to the low number of Botnet instances that were available to train the model as well as the difference between Data Exfiltration and the other datasets regarding the TCP state. In a Support Vector algorithm, the features are leveraged to calculate where a separation is drawn between the classification of Botnet or normal traffic is made. One selection of features may result in the Support Vector that separates classifications of Botnet and normal traffic being drawn such that there is a higher number of false positives while another choice of feature selection may result in a high number of false negatives.

Figure 31. Support Vector Performance Summary

P. SUPPORT VECTOR ON SERVICE SCAN

Figure 32 shows that the Support Vector algorithm consistently achieved greater than 99.83% accuracy, precision, and recall against the Service Scan dataset. Regardless of the selection of features, the performance as measured by accuracy, precision, and recall remained largely unchanged.

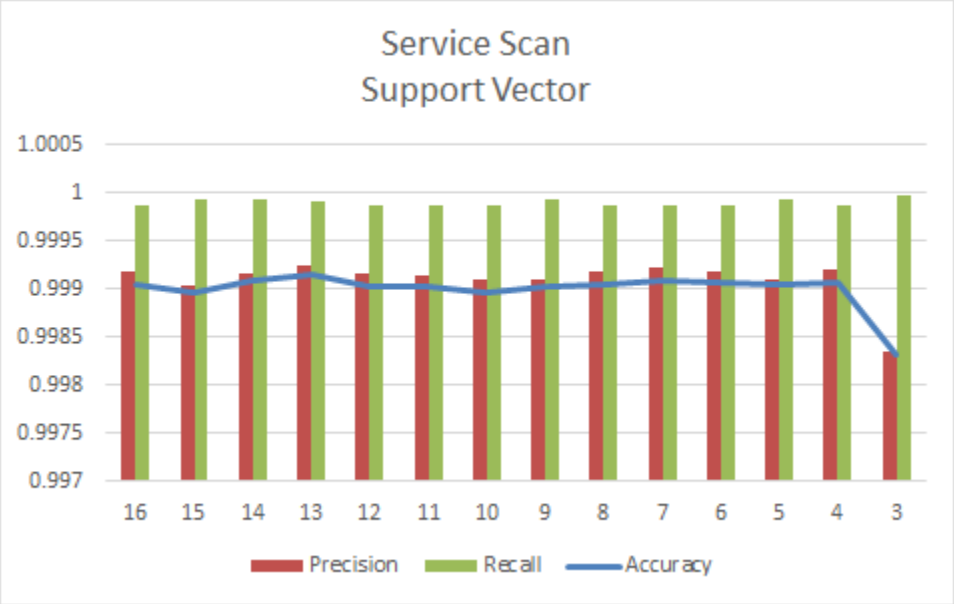


Figure 32. Support Vector Service Scan Performance Summary

Q. SUPPORT VECTOR ON OS FINGERPRINTING

Figure 33 shows that the Support Vector algorithm consistently achieved greater than 99.69% accuracy, precision, and recall against the Service Scan dataset. Regardless of the selection of features, the performance as measured by accuracy, precision, and recall remained largely unchanged.

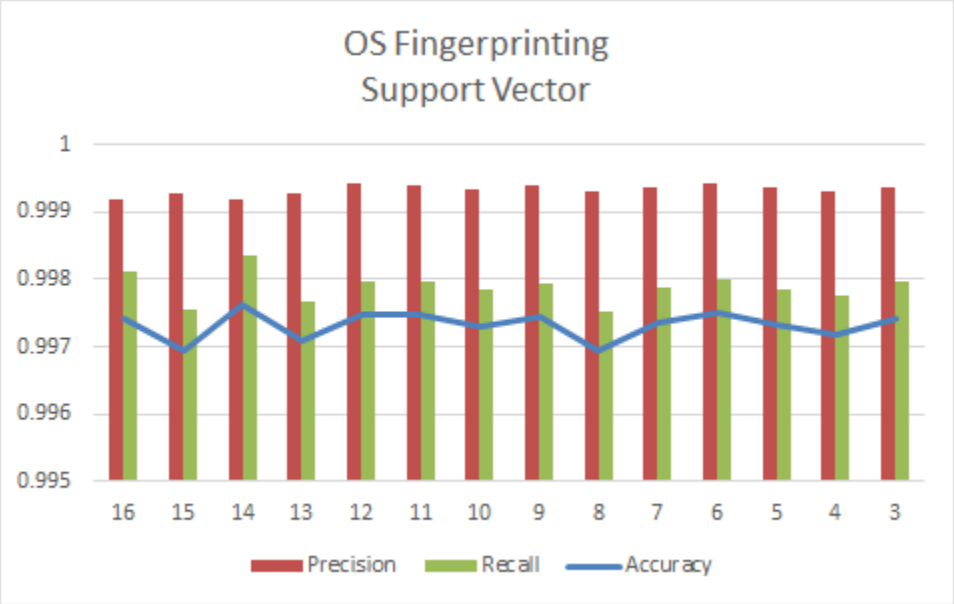


Figure 33. Support Vector OS Fingerprinting Performance Summary

R. SUPPORT VECTOR ON KEYLOGGING

Figure 34 shows that while performance as measured by accuracy is consistently greater than 98.5%, Support Vector results were widely inconsistent and varied in precision and recall (varying from 88% and up) through each iteration of feature selection. As the number of features were reduced, precision and recall alternated in results, as one metric improved the other would decrease. This performance can best be explained by the selection of features changing where the line of separation between Botnet and normal traffic is made. The iteration of feature selection caused the Support Vector that determines the separation between Botnet and normal traffic to oscillate between having a higher number of false positives and having a higher number of false negatives.

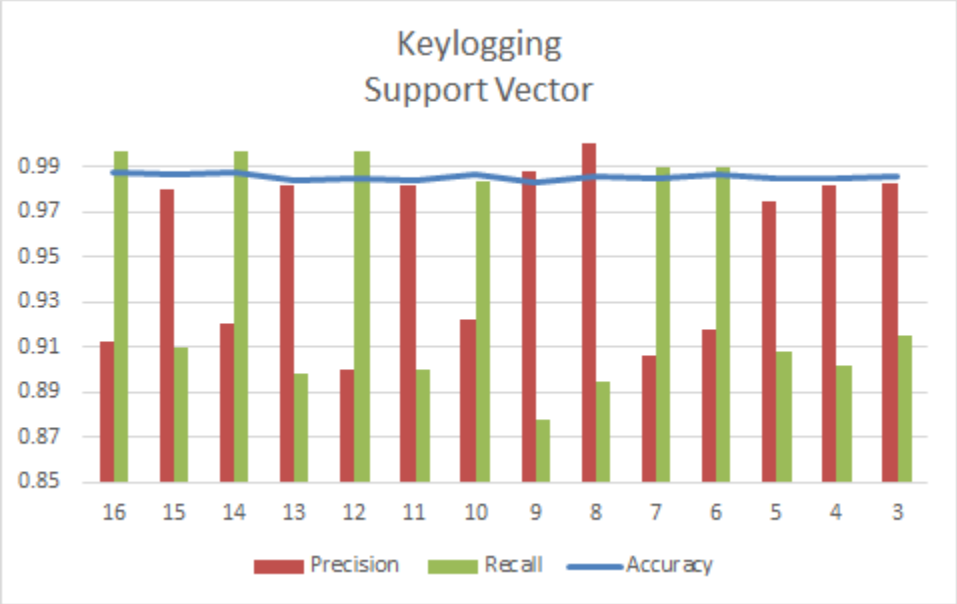


Figure 34. Support Vector Keylogging Performance Summary

S. SUPPORT VECTOR ON DATA EXFILTRATION

At first glance, the performance of Support Vector on the Data Exfiltration dataset shown in Figure 35 is deceiving. While performance as measured by accuracy is consistently greater than 99.5%, precision is very inconsistent with values ranging between 46.1% and 100%, and the value for recall never exceeds 46.6%. This is likely due to the small amount of Data Exfiltration traffic in the dataset on which the algorithm could be trained. There were only 118 instances of Data Exfiltration traffic in the dataset compared to 14,484 instances of normal traffic. If an algorithm were to classify all instances in the Data Exfiltration dataset as normal, then it would still achieve an accuracy of just under 99.2%. However, in that same notional scenario the recall would be 0%. Precision is undefined in this notional scenario because it is zero true positives divided by the sum of zero false positives and zero true positives. This is illustrated in Table 4.

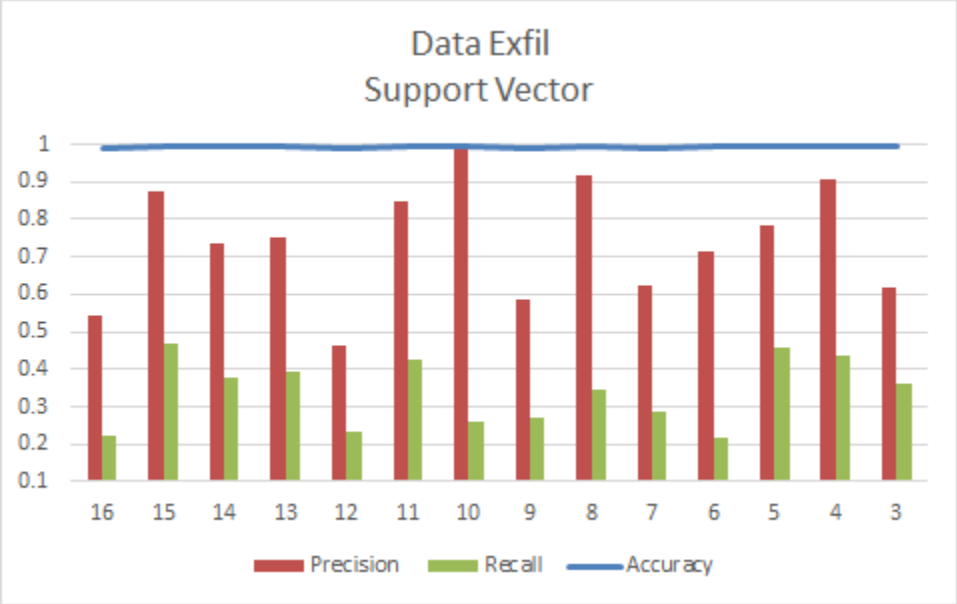


Figure 35. Support Vector Data Exfiltration Performance Summary

Table 4. Notional Data Exfiltration Accuracy Matrix

	TN	FP	FN	TP	Accuracy	Precision	Recall
Data Exfil	14484	0	118	0	0.991919	Undefined	0

T. SUMMARY OF ANALYSIS

The analysis conducted during this research focused on the performance of Machine Learning models and how model performance is impacted by feature selection and the quantity and quality of data. Built-in Python tools from Scikit-Learn modules were used to calculate the importance of the features relative to each dataset. The team was able to demonstrate that feature importance varies between datasets, which may affect the optimization of ML model performance and the ability to successfully detect Botnet traffic. This research found that some traffic types were consistently easier to detect. The order of the datasets by performance from best to worst were Service Scan, OS Fingerprinting, Keylogging, and Data Exfiltration. Of note, the k-Nearest Neighbor model exhibited a

notable pattern of behavior against the Data Exfiltration dataset in which a local maximum was reached during the sequence of feature reduction. No other scenario exhibited this behavior. Additionally, analysis showed a positive correlation between model performance against a given dataset and the prevalence of Botnet traffic. Finally, analysis revealed that different models performed better overall against some datasets versus others. From best overall performance to worst performance, the order of the models are Random Forest, k-Nearest Neighbor, and Support Vector.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS

A. SUMMARY OF RESEARCH

The primary objective of this research was to investigate the application of Machine Learning (ML) to ARGUS data to ultimately answer three research questions regarding the broader application of ML for network security. The three research questions are: (1) what operations and maintenance considerations apply to long term utilization of ML algorithms against Botnet traffic; (2) which, if any, existing Machine Learning algorithms or techniques could be usefully applied to analyze network flow data: and (3) what caveats/ limitations must be considered by consumers of alerts produced by ML analysis of network flow data?

1. Research Question 1

The first research question, what operations and maintenance considerations apply to long term utilization of ML algorithms against Botnet traffic, shaped the perspective of this research towards long term ML for use in defense of the DoDIN. This research highlighted two long-term ML utilization considerations regarding operations and maintenance for the detection of Botnet traffic. The first consideration that was identified for an ML solution identified was the dataset used to train the ML models. The comparison between the results of this research and the results of the previous BOT-IoT research as shown in Table 5 indicated that the ideal approach to detect Botnets or otherwise malicious traffic was to train the ML model to detect a specific type of traffic rather than several types of traffic at the same time. This is important because ML models learn to distinguish between the data presented. However, if one were to attempt to train the ML model to detect and distinguish between multiple types of traffic it becomes significantly less effective. In the case of this research, the quality and size of the dataset were determined to be important. The datasets were shaped by two critical factors, the quality of the data and the quantity of data. In terms of quality, the dataset must properly represent the TTPs employed for that type of traffic because ML model's performance is a reflection of the quality of data on which it was trained. For example, if an ML model is trained with data

to classify all red and round objects as apples it will misclassify anything red and round that is not an apple. The normal traffic must also be representative of typical internet traffic seen in a NOC environment. In terms of volume, the larger datasets containing relevant instances of each traffic to be made available to the ML model for training increased the effectiveness the models' overall performance. The characteristics size and quality that shaped the datasets used to train ML models were highly influential in the results of this research.

Table 5. Random Forest Model Performance against Each Data Type

		Random Forest			Bot-IoT Paper		
		Accuracy	Precision	Recall	Accuracy	Precision	Recall
Service Scan	Average	0.999818	0.999867	0.999951			
	High	0.999919	0.999952	0.999971	0.995682	0.998590	0.997062
	Low	0.999552	0.999627	0.999924			
OS Fingerprinting	Average	0.999680	0.999903	0.999764			
	High	0.999973	1.000000	0.999986	0.991735	0.998429	0.993078
	Low	0.998699	0.999553	0.999093			
Keylogging	Average	0.998022	0.993812	0.991149			
	High	0.999546	1.000000	1.000000	0.987273	0.985294	0.917808
	Low	0.995915	0.979661	0.979094			
Data Exfil	Average	0.999291	0.981329	0.928372			
	High	1.000000	1.000000	1.000000	0.987578	0.000000	0.000000
	Low	0.998631	0.913043	0.880000			

This table compares the results of Random Forest against each dataset as compared to the results on those same Botnet traffic types as reported by the authors of the Bot-IoT Paper. The Bot-IoT paper lists scores of zero for precision and recall.

The second consideration of this research for long-term ML utilization was the selection of features within each dataset used to train each ML model. Each model's features had to be tailored for the type of traffic targeted for detection and for the model used for detection. This research showed that feature importance and selection are significant and vary for each traffic type and each ML model. However, the features that were selected for the datasets in this research may not yield the same results on a dataset that is more representative of realistic Botnet and normal traffic. Therefore, ranking by feature importance should be done on the dataset followed by feature selection being performed for the model.

The quality and quantity of data shape the management and accumulation of datasets to be used to train ML models. While this point will be emphasized in greater depth, careful management and strict filtering must be considered when collecting and organizing datasets for use in ML models for network defense. Data can quickly become obsolete in the modern cyber environment, sustained efforts to maintain an up-to-date database of existing threat traffic types is paramount system maintenance and operational success.

2. Research Question 2

The second research question, which, if any, existing Machine Learning algorithms or techniques could be usefully applied to analyze network flow data, became self-evident during the analysis of our findings. There is no "best" Machine Learning model that can be applied universally to all datasets, but given a particular dataset there could be an ideal model. Between the ML algorithms explored over the course of this research, Random Forest consistently outperformed the other models. Random Forest proved to be less computationally complex. Computational complexity closely ties to the demand for computational resources, the more complex the higher the resource demand. The characteristic serves as an attractive quality in resource constrained environments which should be weighed in any decision-making process for the selection of an ML solution for network security.

To offer perspective, this research was accomplished using a COTS computer with 16 gigabytes of memory, a 4-gigahertz processor, and a 4-gigabyte graphics card. While computational complexity and resource demand can be somewhat mitigated by increasing the capabilities of a computer or distributing the workload over multiple devices, Random Forest starkly contrasted the other models by more quickly and accurately classifying the datasets for each traffic type.

3. Research Question 3

The third research question, what caveats/limitations must be considered by consumers of alerts produced by ML analysis of network flow data, presented noteworthy items of consideration discovered over the course of this research. While the Random

Forest algorithm performed well there was always a risk of false positives and false negatives which was demonstrated in Chapter IV. The risk of false positives contributed to the discussion regarding the question of what caveats/limitations must be considered by consumers of alerts produced by Machine Learning analysis of network flow data. This risk may be increased once an ML model has been trained using datasets composed of more realistic examples of network traffic. One way to mitigate the impact of that risk would be to incorporate context to the results of ML models by comparing traffic that is typically discovered earlier vs. later in an Offensive Cyber Campaign (OCO). OCOs leverage various methods and techniques that lie beyond the scope of this discussion, but for simplicity's sake we will surmise that regarding the detection of certain types of traffic, the order of detection matters. For example, an advanced persistent threat (APT) will make some effort to enumerate the target network by using information gathering tactics. At some point during the OCO, the APT may use information gained about the target network to conduct some action such as exfiltrating data. In this example, the context of the type of traffic being detected matters. So, if the operator or system defending the network detects Service Scan or OS Fingerprinting followed by data exfiltration against the same network addresses, then the detection is much less likely to be a false positive. It is important to recognize that this information could be used against the defender. If the dataset used to train the ML model is not tightly controlled, then the APT could corrupt the dataset in an attempt to circumvent detection. The risk of this scenario would be the highest in an environment where the ML model is being trained using live traffic.

The three research questions explored over the course of this research culminate in enabling us to answer an overarching question pertaining to network security. Can ML using ARGUS data be used to detect Botnets for network defense and security. The results of experimentation and analysis conducted over the course of the research support the hypothesis that ML can accurately and reliably use ARGUS data to detect the presence of Botnets and traffic representative of Botnet traffic. The ML algorithms tested each achieved various levels of performance from perfect detection and classification to unreliable. We can conclude that when appropriate models are selected and properly trained, ML can serve as an early warning system for Botnet threats to a network and to

automate elements of the detection process to better support network operators. More experimentation and testing are warranted before implementing new tactics and techniques to defend the DoDIN, some suggestions are outlined in the Future Work section of this chapter.

B. FUTURE WORK

1. Dataset Creation

Datasets could be created for each broad type of malicious traffic. They should be balanced between normal and malicious traffic and should also be as representative of each traffic type as possible. Including a broad range of realistic traffic according to type may aid future ML models in being able to detect malicious traffic in operational environments. One method of achieving such results would be to include more stealthy attack tactics and reduce the overall number of both overt and DoS attacks in the training and test datasets.

The production of datasets may be an ongoing and evolving endeavor. As understanding of the cyber domain evolves, so will the techniques and tactics implemented by nefarious actors. ML algorithms are susceptible to variations in the data which may lead to misclassification of network flow data. For example, if the model continually learns based upon live traffic inputs, a malicious actor may contaminate the training data by incrementally injecting traffic that is more representative of the malicious traffic or exploitation method that the actor intends to send. Essentially, the malicious actor desensitizes the ML model to the malicious traffic. Therefore, DOD must maintain tight control over how models are trained.

2. Model Optimization

Each of the models explored during this research have parameters that can be optimized to increase performance. Due to time limitations, the optimization of those parameters could not be fully explored. Once realistic datasets become available, further research is needed to identify the proper parameter values for each model on each dataset. For example, research could be conducted to evaluate the results of a Support Vector using a different kernel or evaluate a Random Forest with decision trees of varying breadth and

depth. Exploration of alternate kernels or variations to the algorithms may yield more favorable results, particularly for models that initially fail to achieve acceptable results.

3. Model Testing in a Training Environment

Once the datasets have been produced as recommended and the parameters for each model have been optimized, the models need to be tested in a training environment. Since the code that was developed for this research is readily available for use towards the efforts of follow-on research, moving from an academic to a lab environment for testing is feasible in the immediate future. The idea structure for testing based upon the scope of this research would be for a Red Team to conduct a cyber campaign against a target in the training environment. Packet Captures of the traffic from the Red Team would be converted to an ARGUS comma separated value file, then that file would be processed by a trained ML model to see whether the Red Team's malicious traffic is properly classified.

4. Neural Network Exploration

Due to time constraints, exploration of the use of neural networks in the context of this research was not feasible. Neural networks may serve as a viable option in the use of ML to detect Botnets.

C. LIMITATIONS

1. Time

The single greatest limitation to this research was time, due to the structure of the academic experience at the Naval Postgraduate school, students must set attainable goals to be accomplish during their limited tenure. As a result, initial ambitions to develop a user interface and conduct a user study were quickly tapered to focus on the most important aspects of ML utilizing ARGUS data.

2. Knowledge

The research process in pursuing ML applications was as much a desire to learn as it was to meet graduation requirements. The research process taught our team a great deal about ML, coding in Python, and cyber threats. With more time and the knowledge our

team gained over the course of this research we could have accomplished a great deal more in this area of study.

3. Dataset

While the BOT-IoT dataset provided our research team with an exceptional way to conduct our research by having ARGUS data in a convenient format, the dataset was limited in quantity of Keylogging and Data exfiltration training data. This is a limiting factor because as described earlier in the paper, effectiveness of the models was highly influenced by the quality and quantity of the data.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Argus. (n.d.). *ARGUS + ML*. https://openargus.org/argus-ml

Agazzi, A. E. (2020, July 6). *Smart home, security concerns of IOT.* Retrieved from arXiv.org: https://arxiv.org/abs/2007.02628

Arlitt, M. &. (2005). An analysis of TCP reset behaviour on the internet. *ACM SIGCOMM Computer Communications Review, 35*(1), 37–44.

Bellman, R. &. (1957). On the role of dynamic programming in statistical communication theory. *IRE Transactions on Information Theory, 3*(3), 197–203.

Chen, X. L. (2017). Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv Preprint arXiv:1712.05526.*

Chowdhury, S., Khanzadeh, M., Akula, R., Zhang, F., Zhang, S., Medal, H., . . . Bian, L. (2017). Botnet detection using graph-based feature clustering. *Journal of Big Data*, *4*(14). https://doi.org/10.1186/s40537-017-0074-7

DARPA. (1981, September). *Transmission control protocol. RFC 793*. Retrieved from IETF Tools: https://tools.ietf.org/html/rfc793

Ding, S. (2018). Machine learning for cybersecurity: Network-based botnet detection using time-limited flows. *Caltech Undergrad Research Journal*, 2018 issue.

Garcia, S. G. (2014). An empirical comparison of botnet detection methods. *Computers & Security, 45*, 100–123.

Han, S. (2003). Individual adoption of information systems in organisations: a literature review of technology acceptance model. *TUCS Technical Report 540.* TUCS.

IBM Security. (2016, March). *The inside story on botnets*. Retrieved from ibm.com: https://www.ibm.com/downloads/cas/V3YJVYZX

Khattak, S., Ramay, N. R., Khan, K. R., Syed, A. A., & Khayam, A. S. (2014). A taxonomy of botnet behavior, detection, and defense. *IEEE Communications Surveys & Tutorials*, 898–924.

Knecht, T. (2016, September 21). *A brief history of bots and how they've shaped the internet today*. Retrieved from Abusix: https://www.abusix.com/blog/a-brief-history-of-bots-and-how-theyve-shaped-the-internet-today

Koroniotis, N. M. (2019). Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems*, 779–796.

Kotsiantis, S. K. (2006). Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering, 30(1)*, 25–36.

Ross, J. (2020). *Training-ML-models-to-detect-malicious-network-traffic*. Retrieved from GitHub: https://github.com/jtrii0211/Training-ML-models-to-detect-malicious-network-traffic/blob/main/Thesis.py

Saad, S., Traore, I., Ghorbani, A., Sayed, B., Zhao, D., Lu, W., & ... Hakimian, P. (2011). Detecting P2P botnets through network behavior analysis and machine learning. *In 2011 Ninth Annual International Conference on Privacy, Securty and Trust* (pp. 174–180). IEEE.

Salazar, D. (2018). *Leveraging maching-learning to enhance network security.* [Master's thesis, Naval Postgraduate School]. NPS Archive: Calhoun. http://hdl.handle.net/10945/59578

Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. *2010 IEEE Symposium on Security and Privacy*, 305–316.

Weaver, N., Sommer, R, & Paxson, V. (2009, February). *Detecting forged TCP reset packets.* NDSS.

Zhao, D. T. (2013). Botnet detection based on traffic behavior analysis and flow intervals. *Computers & Security, 39*, 2–16.

Zhao, D., Traore, I., Ghorbani, A., Sayed, B., & Saad, S. (2012). Peer to peer botnet detection based on flow intervals. *In IFIP International Information Security Conference* (pp. 87–102). Springer, Berlin, Heidelberg.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California